

Contents

[Azure Blob Storage documentation](#)

[Overview](#)

[What is Azure Blob Storage?](#)

[Compare core storage services](#)

[Blob Storage feature support](#)

[Blob storage](#)

[Overview](#)

[Introduction to Blob storage](#)

[Quickstarts](#)

[Work with blobs](#)

[Azure portal](#)

[Storage Explorer](#)

[PowerShell](#)

[CLI](#)

[.NET](#)

[.NET \(v12 SDK\)](#)

[.NET \(v11 SDK\)](#)

[Java](#)

[Java \(v12 SDK\)](#)

[Java \(v8 SDK\)](#)

[Spring](#)

[Python](#)

[Python \(v12 SDK\)](#)

[Python \(v2.1 SDK\)](#)

[JavaScript](#)

[JavaScript for Node.js](#)

[JavaScript for browser](#)

[C++ \(v12 SDK\)](#)

[Go](#)

[PHP](#)

[Ruby](#)

[Xamarin](#)

Tutorials

[Upload and process image files](#)

[.NET](#)

[JavaScript](#)

[Migrate data to cloud with AzCopy](#)

[Optimize storage application performance and scalability](#)

[1 - Create a VM and storage account](#)

[2 - Upload large data to an Azure storage account](#)

[3- Download large data from an Azure storage account](#)

[4 - Verify throughput and latency metrics](#)

[Host a static website](#)

[Design applications for high availability](#)

[1 - Make your application highly available](#)

[2 - Simulate a failure in reading data from the primary region](#)

[Add role assignment conditions](#)

[Portal](#)

[PowerShell](#)

[CLI](#)

Samples

[.NET](#)

[Java](#)

[Python](#)

[JavaScript](#)

[C++](#)

[Other languages](#)

[Azure PowerShell](#)

[Azure CLI](#)

[Overview](#)

[Retrieve/rotate the access keys](#)

- Calculate size of Blob storage container
- Delete containers with a specific prefix
- Azure Resource Graph queries

Concepts

- Storage accounts

- Overview

- Premium performance

- Authorization

- Authorizing data operations

- Authorize with Azure AD

- Authorize with Azure roles

- Authorize with conditions

- Actions and attributes for conditions

- Security for conditions

- Example conditions

- Authorize with Shared Key

- Delegate access with shared access signatures (SAS)

- Authorizing management operations

Security

- Security recommendations

- Security baseline

- Azure Storage encryption at rest

- Encryption with customer-managed keys

- Encryption on the request with customer-provided keys

- Encryption scopes

- Use Azure Private Endpoints

- Configure network routing preference

Data protection

- Overview

- Soft delete for containers

- Soft delete for blobs

- Versioning

[Point-in-time restore](#)

[Snapshots](#)

[Change feed](#)

[Immutable storage for blobs](#)

[Overview](#)

[Time-based retention policies](#)

[Legal holds](#)

[Redundancy and disaster recovery](#)

[Data redundancy](#)

[Customer-managed failover for disaster recovery](#)

[Blob access tiers and lifecycle management](#)

[Overview](#)

[Blob rehydration from Archive tier](#)

[Lifecycle management policies](#)

[Object replication](#)

[Performance, scaling, and cost optimization](#)

[Performance and scalability checklist](#)

[Latency in Blob storage](#)

[Azure Storage reserved capacity](#)

[Plan and manage costs](#)

[Scalability and performance targets](#)

[Blob storage](#)

[Standard storage accounts](#)

[Premium block blob storage accounts](#)

[Premium page blob storage accounts](#)

[Storage resource provider](#)

[Find, search, and understand blob data](#)

[Blob inventory](#)

[Blob index tags](#)

[Full text search](#)

[Azure Cognitive Search](#)

[Data migration](#)

- [Storage migration overview](#)
- [Compare data transfer solutions](#)
 - [Large dataset, low network bandwidth](#)
 - [Large dataset, moderate to high network bandwidth](#)
 - [Small dataset, low to moderate network bandwidth](#)
 - [Periodic data transfer](#)
- [Monitoring](#)
 - [Monitor Blob storage](#)
 - [Transition from classic metrics](#)
 - [Monitoring \(classic\)](#)
 - [Storage Analytics](#)
 - [Metrics](#)
 - [Logs](#)
- [Protocol support](#)
 - [NFS 3.0](#)
 - [Overview](#)
 - [Performance considerations](#)
 - [Limitations and known issues](#)
 - [SFTP](#)
 - [Overview](#)
 - [Performance considerations](#)
 - [Limitations and known issues](#)
 - [Event handling](#)
 - [Page blob features](#)
 - [Static websites](#)
 - [Upgrading to Data Lake Storage Gen2](#)
- [How to](#)
 - [Configure Blob Storage](#)
 - [Create and manage storage accounts](#)
 - [Create a storage account](#)
 - [Upgrade a storage account](#)
 - [Recover a deleted storage account](#)

- [Get account configuration properties](#)
- [Move across regions](#)
- [Upgrade to Data Lake Storage Gen2](#)
- [Create and manage containers](#)
 - [Manage containers \(Azure portal\)](#)
 - [Manage containers \(CLI\)](#)
 - [Manage containers \(PowerShell\)](#)
- [Create and manage blobs](#)
 - [Manage blobs \(CLI\)](#)
 - [Manage blobs \(PowerShell\)](#)
 - [Use blob index tags](#)
 - [Enable blob inventory reports](#)
 - [Calculate blob count and total size](#)
- [Authorize access to blob data](#)
 - [Authorization options for users](#)
 - [Portal](#)
 - [PowerShell](#)
 - [Azure CLI](#)
 - [Manage access rights with Azure RBAC](#)
 - [Authorize with Shared Key](#)
 - [View and manage account keys](#)
 - [Configure connection strings](#)
 - [Use the Azure Storage REST API](#)
 - [Prevent authorization with Shared Key](#)
 - [Delegate access with shared access signatures \(SAS\)](#)
 - [Create a user delegation SAS](#)
 - [Create a service SAS](#)
 - [Create an account SAS \(.NET\)](#)
 - [Define a stored access policy](#)
 - [Create a SAS expiration policy](#)
 - [Manage anonymous read access to blob data](#)
 - [Configure anonymous read access for containers and blobs](#)

Prevent anonymous read access to blob data

Access public containers and blobs anonymously (.NET)

Secure blob data

Manage Azure Storage encryption

Check whether a blob is encrypted

Manage encryption keys for the storage account

Provide an encryption key on a request

Manage encryption scopes

Enable infrastructure encryption for the account

Configure client-side encryption

Configure network security

Require secure transfer

Configure firewalls and virtual networks

Manage Transport Layer Security (TLS)

Configure network routing preference

Enable threat protection with Microsoft Defender for Storage

Protect blob data

Lock a storage account

Enable soft delete for containers

Enable blob versioning

Enable and manage soft delete for blobs

Enable blob soft delete

Manage and restore soft-deleted blobs

Enable point-in-time restore

Create snapshots (.NET)

Process change feed logs

Configure an immutability policy

Configure version-level immutability policies

Configure container-level immutability policies

Manage redundancy and failover

Change redundancy configuration

Design highly available applications

- [Check the Last Sync Time property](#)
- [Initiate account failover](#)
- [Manage blob tiering and lifecycle](#)
 - [Change a blob's access tier](#)
 - [Manage data archiving](#)
 - [Archive a blob](#)
 - [Rehydrate an archived blob](#)
 - [Handle an event on blob rehydration](#)
 - [Configure lifecycle management policies](#)
- [Manage object replication](#)
 - [Configure object replication policies](#)
 - [Prevent object replication across tenants](#)
- [Manage concurrency](#)
- [Host a static website](#)
 - [Host a static website](#)
 - [Integrate with Azure CDN](#)
 - [Use GitHub Actions to deploy a static site to Azure Storage](#)
- [Use a custom domain](#)
- [Route events to a custom endpoint](#)
- [Transfer data](#)
- [AzCopy](#)
 - [Get started](#)
 - [Authorize with Azure AD](#)
 - [Optimize performance](#)
 - [Use logs to find errors and resume jobs](#)
 - [Examples: Upload](#)
 - [Examples: Download](#)
 - [Examples: Copy between accounts](#)
 - [Examples: Synchronize](#)
 - [Examples: Properties and metadata](#)
 - [Examples: Amazon S3 buckets](#)
 - [Examples: Google Cloud Storage buckets](#)

Troubleshoot AzCopy issues

BlobFuse

[What is BlobFuse2?](#)

[Deploy BlobFuse2 on Linux](#)

[Deploy BlobFuse v1 on Linux](#)

[Troubleshooting](#)

Azure Data Factory

[Transfer data by using SFTP](#)

[Mount storage by using NFS](#)

[Mount storage from Linux using BlobFuse](#)

[Transfer data with the Data Movement library](#)

Develop applications

[Set up Azure AD](#)

[Get an access token for authorization with Azure AD](#)

[Authorize from an application running in Azure](#)

[Authorize from a native or web application](#)

Write code

[.NET](#)

[Get started](#)

[Work with containers](#)

[Work with blobs](#)

[JavaScript](#)

[Get started](#)

[Work with SAS tokens](#)

[Work with containers](#)

[Work with blobs](#)

[Test with a storage emulator](#)

[Use the Azurite open-source emulator](#)

[Use Azurite to run automated tests](#)

[Use the Azure Storage emulator \(deprecated\)](#)

Monitor

[Scenarios and best practices](#)

[Use Storage insights](#)

[Troubleshoot](#)

[Performance](#)

[Availability](#)

[Client errors](#)

[Monitor \(classic\)](#)

[Enable and manage metrics \(classic\)](#)

[Enable and manage logs \(classic\)](#)

[Troubleshoot](#)

[Monitor, diagnose, and troubleshoot](#)

[Troubleshoot Latency issues](#)

[Reference](#)

[Blob Storage API reference](#)

[AzCopy v10](#)

[Configuration settings](#)

[azcopy](#)

[azcopy bench](#)

[azcopy copy](#)

[azcopy doc](#)

[azcopy env](#)

[azcopy jobs](#)

[azcopy jobs clean](#)

[azcopy jobs list](#)

[azcopy jobs remove](#)

[azcopy jobs resume](#)

[azcopy jobs show](#)

[azcopy list](#)

[azcopy login](#)

[azcopy login status](#)

[azcopy logout](#)

[azcopy make](#)

[azcopy remove](#)

[azcopy sync](#)
[azcopy set-properties](#)
[BlobFuse2](#)
 [Configuration settings](#)
 [Commands](#)
 [The BlobFuse2 command set](#)
 [BlobFuse2 mount](#)
 [BlobFuse2 mount all](#)
 [BlobFuse2 mount list](#)
 [BlobFuse2 mountv1](#)
 [BlobFuse2 unmount](#)
 [BlobFuse2 unmount all](#)
 [BlobFuse2 completion](#)
 [BlobFuse2 completion bash](#)
 [BlobFuse2 completion fish](#)
 [BlobFuse2 completion PowerShell](#)
 [BlobFuse2 completion zsh](#)
 [BlobFuse2 secure](#)
 [BlobFuse2 secure encrypt](#)
 [BlobFuse2 secure decrypt](#)
 [BlobFuse2 secure get](#)
 [BlobFuse2 secure set](#)
 [BlobFuse2 version](#)
 [BlobFuse2 help](#)
[Resource Manager template](#)
[Monitoring data](#)
[Host keys \(SFTP\) support](#)
[Azure Policy built-ins](#)
[Resources](#)
 [Azure updates](#)
 [Azure Storage Explorer](#)
 [Download Storage Explorer](#)

- [Get started with Storage Explorer](#)
- [Sign in to Storage Explorer](#)
- [Storage Explorer networking](#)
- [Storage Explorer release notes](#)
- [Troubleshoot Storage Explorer](#)
- [Connect an emulator to Storage Explorer](#)
- [Storage Explorer command line options](#)
- [Storage Explorer direct link](#)
- [Storage Explorer security](#)
- [Storage Explorer soft delete](#)
- [Storage Explorer blob versioning](#)
- [Storage Explorer manage Azure Blob storage](#)
- [Storage Explorer create file shares](#)
- [Storage Explorer support policy and lifecycle](#)
- [Storage Explorer accessibility](#)
- [Blob storage on Microsoft Q&A](#)
- [Blob storage on Stack Overflow](#)
- [Pricing for block blobs](#)
- [Pricing for page blobs](#)
- [Azure pricing calculator](#)
- [Videos](#)
 - [Compare access with NFS to Azure Blob Storage, Azure Files, and Azure NetApp Files](#)
- [NuGet packages \(.NET\)](#)
 - [Microsoft.Azure.Storage.Common \(version 11.x\)](#)
 - [Azure.Storage.Common \(version 12.x\)](#)
 - [Microsoft.Azure.Storage.Blob \(version 11.x\)](#)
 - [Azure.Storage.Blob \(version 12.x\)](#)
 - [Azure Configuration Manager](#)
 - [Azure Storage Data Movement library](#)
 - [Storage Resource Provider library](#)
- [Source code](#)
- [.NET](#)

Azure Storage client library

[Version 12.x](#)

[Version 11.x and earlier](#)

[Data Movement library](#)

[Storage Resource Provider library](#)

[Java](#)

[Azure Storage client library version 12.x](#)

[Azure Storage client library version 8.x and earlier](#)

[Node.js](#)

[Azure Storage client library version 12.x](#)

[Azure Storage client library version 10.x](#)

[Python](#)

[Azure Storage client library version 12.x](#)

[Azure Storage client library version 2.1](#)

[Compliance offerings](#)

Data Lake Storage Gen2

[Switch to Data Lake Storage Gen1 documentation](#)

[Overview](#)

[Introduction to Data Lake Storage](#)

[Blob Storage feature support](#)

[Tutorials](#)

[Use with Synapse SQL](#)

[Use with Databricks and Spark](#)

[Use with Apache Hive and HDInsight](#)

[Use with Databricks Delta and Event Grid](#)

[Use with other Azure services](#)

[Concepts](#)

[Best practices](#)

[Query acceleration](#)

[Premium tier for Data Lake Storage](#)

[Architecture](#)

[Azure Blob File System driver for Hadoop](#)

[Azure Blob File System URI](#)

[About hierarchical namespaces](#)

[Security](#)

[Security recommendations](#)

[Access control model](#)

[Access control lists](#)

[Compatibility](#)

[Multi-protocol access](#)

[Supported Blob storage features](#)

[Supported Azure services](#)

[Supported open source platforms](#)

[Monitoring](#)

[Monitor Blob storage](#)

[Transition from classic metrics](#)

[Monitoring \(classic\)](#)

[Storage Analytics](#)

[Metrics](#)

[Logs](#)

[Monitor, diagnose, and troubleshoot](#)

[How to](#)

[Create a storage account](#)

[Migrate accounts and data stores](#)

[Migrate from Data Lake Storage Gen1](#)

[Migrate using Azure portal](#)

[Migration guidelines and patterns](#)

[Migrate HDFS stores](#)

[Migrate an HDFS store offline](#)

[Migrate an HDFS store online](#)

[Transfer data](#)

[AzCopy](#)

[Get started](#)

[Authorize with Azure AD](#)

Optimize performance

Use logs to find errors and resume jobs

Examples: Upload

Examples: Download

Examples: Copy between accounts

Examples: Synchronize

Examples: Properties and metadata

Examples: Amazon S3 buckets

Examples: Google Cloud Storage buckets

Azure Data Factory

Transfer data with the DistCp tool

Transfer data with the Data Movement library

Manage access control

Azure portal

Storage Explorer

PowerShell

CLI

.NET

Java

Python

JavaScript

Work with data

Storage Explorer

PowerShell

CLI

.NET

Java

Python

JavaScript

Query acceleration

Hadoop File System CLI

Use with Azure services

Reference

[.NET](#)

[Java](#)

[Python](#)

[JavaScript](#)

[C++](#)

[REST](#)

[Azure CLI](#)

[Query acceleration reference](#)

[AzCopy v10](#)

[azcopy](#)

[azcopy bench](#)

[azcopy copy](#)

[azcopy doc](#)

[azcopy env](#)

[azcopy jobs](#)

[azcopy jobs clean](#)

[azcopy jobs list](#)

[azcopy jobs remove](#)

[azcopy jobs resume](#)

[azcopy jobs show](#)

[azcopy list](#)

[azcopy login](#)

[azcopy logout](#)

[azcopy make](#)

[azcopy remove](#)

[azcopy sync](#)

[azcopy set-properties](#)

[Resource Manager template](#)

[Azure Policy built-ins](#)

[Resources](#)

[Known issues](#)

[Azure Roadmap](#)

[Azure updates](#)

[Azure Storage client tools](#)

[Azure Storage Explorer](#)

[Download Storage Explorer](#)

[Get started with Storage Explorer](#)

[Storage Explorer release notes](#)

[Troubleshoot Storage Explorer](#)

[Storage Explorer security](#)

[Storage Explorer blob versioning](#)

[Storage Explorer manage Azure Blob storage](#)

[Storage Explorer create file shares](#)

[Storage Explorer support policy and lifecycle](#)

[Storage Explorer accessibility](#)

[Microsoft Q&A question page](#)

[Azure Storage on Stack Overflow](#)

[Pricing](#)

[Azure pricing calculator](#)

[Pricing calculator](#)

[Service updates](#)

[Stack Overflow](#)

[Videos](#)

[Partners](#)

What is Azure Blob storage?

8/22/2022 • 2 minutes to read • [Edit Online](#)

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 doesn't adhere to a particular data model or definition, such as text or binary data.

About Blob storage

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](#)
- [PHP](#)
- [Ruby](#)

About Azure Data Lake Storage Gen2

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

For more information about Data Lake Storage Gen2, see [Introduction to Azure Data Lake Storage Gen2](#).

Next steps

- [Introduction to Azure Blob storage](#)
- [Introduction to Azure Data Lake Storage Gen2](#)

Introduction to Azure Storage

8/22/2022 • 11 minutes to read • [Edit Online](#)

The Azure Storage platform is Microsoft's cloud storage solution for modern data storage scenarios. Azure Storage offers highly available, massively scalable, durable, and secure storage for a variety of data objects in the cloud. Azure Storage data objects are accessible from anywhere in the world over HTTP or HTTPS via a REST API. Azure Storage also offers client libraries for developers building applications or services with .NET, Java, Python, JavaScript, C++, and Go. Developers and IT professionals can use Azure PowerShell and Azure CLI to write scripts for data management or configuration tasks. The Azure portal and Azure Storage Explorer provide user-interface tools for interacting with Azure Storage.

Benefits of Azure Storage

Azure Storage services offer the following benefits for application developers and IT professionals:

- **Durable and highly available.** Redundancy ensures that your data is safe in the event of transient hardware failures. You can also opt to replicate data across data centers or geographical regions for additional protection from local catastrophe or natural disaster. Data replicated in this way remains highly available in the event of an unexpected outage.
- **Secure.** All data written to an Azure storage account is encrypted by the service. Azure Storage provides you with fine-grained control over who has access to your data.
- **Scalable.** Azure Storage is designed to be massively scalable to meet the data storage and performance needs of today's applications.
- **Managed.** Azure handles hardware maintenance, updates, and critical issues for you.
- **Accessible.** Data in Azure Storage is accessible from anywhere in the world over HTTP or HTTPS. Microsoft provides client libraries for Azure Storage in a variety of languages, including .NET, Java, Node.js, Python, PHP, Ruby, Go, and others, as well as a mature REST API. Azure Storage supports scripting in Azure PowerShell or Azure CLI. And the Azure portal and Azure Storage Explorer offer easy visual solutions for working with your data.

Azure Storage data services

The Azure Storage platform includes the following data services:

- [Azure Blobs](#): A massively scalable object store for text and binary data. Also includes support for big data analytics through Data Lake Storage Gen2.
- [Azure Files](#): Managed file shares for cloud or on-premises deployments.
- [Azure Queues](#): A messaging store for reliable messaging between application components.
- [Azure Tables](#): A NoSQL store for schemaless storage of structured data.
- [Azure Disks](#): Block-level storage volumes for Azure VMs.

Each service is accessed through a storage account. To get started, see [Create a storage account](#).

Review options for storing data in Azure

Azure provides a variety of storage tools and services, including Azure Storage. To determine which Azure technology is best suited for your scenario, see [Review your storage options](#) in the Azure Cloud Adoption Framework.

Sample scenarios for Azure Storage services

The following table compares Files, Blobs, Disks, Queues, and Tables, and shows example scenarios for each.

FEATURE	DESCRIPTION	WHEN TO USE
Azure Files	<p>Offers fully managed cloud file shares that you can access from anywhere via the industry standard Server Message Block (SMB) protocol.</p> <p>You can mount Azure file shares from cloud or on-premises deployments of Windows, Linux, and macOS.</p>	<p>You want to "lift and shift" an application to the cloud that already uses the native file system APIs to share data between it and other applications running in Azure.</p> <p>You want to replace or supplement on-premises file servers or NAS devices.</p> <p>You want to store development and debugging tools that need to be accessed from many virtual machines.</p>
Azure Blobs	<p>Allows unstructured data to be stored and accessed at a massive scale in block blobs.</p> <p>Also supports Azure Data Lake Storage Gen2 for enterprise big data analytics solutions.</p>	<p>You want your application to support streaming and random access scenarios.</p> <p>You want to be able to access application data from anywhere.</p> <p>You want to build an enterprise data lake on Azure and perform big data analytics.</p>
Azure Disks	Allows data to be persistently stored and accessed from an attached virtual hard disk.	<p>You want to "lift and shift" applications that use native file system APIs to read and write data to persistent disks.</p> <p>You want to store data that is not required to be accessed from outside the virtual machine to which the disk is attached.</p>
Azure Queues	Allows for asynchronous message queueing between application components.	<p>You want to decouple application components and use asynchronous messaging to communicate between them.</p> <p>For guidance around when to use Queue storage versus Service Bus queues, see Storage queues and Service Bus queues - compared and contrasted.</p>
Azure Tables	Allow you to store structured NoSQL data in the cloud, providing a key/attribute store with a schemaless design.	<p>You want to store flexible datasets like user data for web applications, address books, device information, or other types of metadata your service requires.</p> <p>For guidance around when to use Table storage versus the Azure Cosmos DB Table API, see Developing with Azure Cosmos DB Table API and Azure Table storage.</p>

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, such as text or binary data.

Blob storage is ideal for:

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

Objects in Blob storage can be accessed from anywhere in the world via HTTP or HTTPS. Users or client applications can access blobs via URLs, the [Azure Storage REST API](#), [Azure PowerShell](#), [Azure CLI](#), or an Azure Storage client library. The storage client libraries are available for multiple languages, including [.NET](#), [Java](#), [Node.js](#), [Python](#), [PHP](#), and [Ruby](#).

For more information about Blob storage, see [Introduction to Blob storage](#).

Azure Files

[Azure Files](#) enables you to set up highly available network file shares that can be accessed by using the standard Server Message Block (SMB) protocol. That means that multiple VMs can share the same files with both read and write access. You can also read the files using the REST interface or the storage client libraries.

One thing that distinguishes Azure Files from files on a corporate file share is that you can access the files from anywhere in the world using a URL that points to the file and includes a shared access signature (SAS) token. You can generate SAS tokens; they allow specific access to a private asset for a specific amount of time.

File shares can be used for many common scenarios:

- Many on-premises applications use file shares. This feature makes it easier to migrate those applications that share data to Azure. If you mount the file share to the same drive letter that the on-premises application uses, the part of your application that accesses the file share should work with minimal, if any, changes.
- Configuration files can be stored on a file share and accessed from multiple VMs. Tools and utilities used by multiple developers in a group can be stored on a file share, ensuring that everybody can find them, and that they use the same version.
- Resource logs, metrics, and crash dumps are just three examples of data that can be written to a file share and processed or analyzed later.

For more information about Azure Files, see [Introduction to Azure Files](#).

Some SMB features are not applicable to the cloud. For more information, see [Features not supported by the Azure File service](#).

Queue storage

The Azure Queue service is used to store and retrieve messages. Queue messages can be up to 64 KB in size, and a queue can contain millions of messages. Queues are generally used to store lists of messages to be processed asynchronously.

For example, say you want your customers to be able to upload pictures, and you want to create thumbnails for each picture. You could have your customer wait for you to create the thumbnails while uploading the pictures.

An alternative would be to use a queue. When the customer finishes their upload, write a message to the queue. Then have an Azure Function retrieve the message from the queue and create the thumbnails. Each of the parts of this processing can be scaled separately, giving you more control when tuning it for your usage.

For more information about Azure Queues, see [Introduction to Queues](#).

Table storage

Azure Table storage is now part of Azure Cosmos DB. To see Azure Table storage documentation, see the [Azure Table Storage Overview](#). In addition to the existing Azure Table storage service, there is a new Azure Cosmos DB Table API offering that provides throughput-optimized tables, global distribution, and automatic secondary indexes. To learn more and try out the new premium experience, see [Azure Cosmos DB Table API](#).

For more information about Table storage, see [Overview of Azure Table storage](#).

Disk storage

An Azure managed disk is a virtual hard disk (VHD). You can think of it like a physical disk in an on-premises server but, virtualized. Azure-managed disks are stored as page blobs, which are a random IO storage object in Azure. We call a managed disk 'managed' because it is an abstraction over page blobs, blob containers, and Azure storage accounts. With managed disks, all you have to do is provision the disk, and Azure takes care of the rest.

For more information about managed disks, see [Introduction to Azure managed disks](#).

Types of storage accounts

Azure Storage offers several types of storage accounts. Each type supports different features and has its own pricing model. For more information about storage account types, see [Azure storage account overview](#).

Secure access to storage accounts

Every request to Azure Storage must be authorized. Azure Storage supports the following authorization methods:

- **Azure Active Directory (Azure AD) integration for blob, queue, and table data.** Azure Storage supports authentication and authorization with Azure AD for the Blob and Queue services via Azure role-based access control (Azure RBAC). Authorization with Azure AD is also supported for the Table service in preview. Authorizing requests with Azure AD is recommended for superior security and ease of use. For more information, see [Authorize access to data in Azure Storage](#).
- **Azure AD authorization over SMB for Azure Files.** Azure Files supports identity-based authorization over SMB (Server Message Block) through either Azure Active Directory Domain Services (Azure AD DS) or on-premises Active Directory Domain Services (preview). Your domain-joined Windows VMs can access Azure file shares using Azure AD credentials. For more information, see [Overview of Azure Files identity-based authentication support for SMB access](#) and [Planning for an Azure Files deployment](#).
- **Authorization with Shared Key.** The Azure Storage Blob, Files, Queue, and Table services support authorization with Shared Key. A client using Shared Key authorization passes a header with every request that is signed using the storage account access key. For more information, see [Authorize with Shared Key](#).
- **Authorization using shared access signatures (SAS).** A shared access signature (SAS) is a string containing a security token that can be appended to the URI for a storage resource. The security token encapsulates constraints such as permissions and the interval of access. For more information, see [Using Shared Access Signatures \(SAS\)](#).
- **Anonymous access to containers and blobs.** A container and its blobs may be publicly available. When you specify that a container or blob is public, anyone can read it anonymously; no authentication is required.

For more information, see [Manage anonymous read access to containers and blobs](#).

Encryption

There are two basic kinds of encryption available for Azure Storage. For more information about security and encryption, see the [Azure Storage security guide](#).

Encryption at rest

Azure Storage encryption protects and safeguards your data to meet your organizational security and compliance commitments. Azure Storage automatically encrypts all data prior to persisting to the storage account and decrypts it prior to retrieval. The encryption, decryption, and key management processes are transparent to users. Customers can also choose to manage their own keys using Azure Key Vault. For more information, see [Azure Storage encryption for data at rest](#).

Client-side encryption

The Azure Storage client libraries provide methods for encrypting data from the client library before sending it across the wire and decrypting the response. Data encrypted via client-side encryption is also encrypted at rest by Azure Storage. For more information about client-side encryption, see [Client-side encryption with .NET for Azure Storage](#).

Redundancy

To ensure that your data is durable, Azure Storage stores multiple copies of your data. When you set up your storage account, you select a redundancy option. For more information, see [Azure Storage redundancy](#).

Transfer data to and from Azure Storage

You have several options for moving data into or out of Azure Storage. Which option you choose depends on the size of your dataset and your network bandwidth. For more information, see [Choose an Azure solution for data transfer](#).

Pricing

When making decisions about how your data is stored and accessed, you should also consider the costs involved. For more information, see [Azure Storage pricing](#).

Storage APIs, libraries, and tools

You can access resources in a storage account by any language that can make HTTP/HTTPS requests. Additionally, Azure Storage offer programming libraries for several popular languages. These libraries simplify many aspects of working with Azure Storage by handling details such as synchronous and asynchronous invocation, batching of operations, exception management, automatic retries, operational behavior, and so forth. Libraries are currently available for the following languages and platforms, with others in the pipeline:

Azure Storage data API and library references

- [Azure Storage REST API](#)
- [Azure Storage client library for .NET](#)
- [Azure Storage client library for Java/Android](#)
- [Azure Storage client library for Node.js](#)
- [Azure Storage client library for Python](#)
- [Azure Storage client library for PHP](#)
- [Azure Storage client library for Ruby](#)
- [Azure Storage client library for C++](#)

Azure Storage management API and library references

- [Storage Resource Provider REST API](#)
- [Storage Resource Provider Client Library for .NET](#)
- [Storage Service Management REST API \(Classic\)](#)

Azure Storage data movement API and library references

- [Storage Import/Export Service REST API](#)
- [Storage Data Movement Client Library for .NET](#)

Tools and utilities

- [Azure PowerShell Cmdlets for Storage](#)
- [Azure CLI Cmdlets for Storage](#)
- [AzCopy Command-Line Utility](#)
- [Azure Storage Explorer](#) is a free, standalone app from Microsoft that enables you to work visually with Azure Storage data on Windows, macOS, and Linux.
- [Azure Resource Manager templates for Azure Storage](#)

Next steps

To get up and running with Azure Storage, see [Create a storage account](#).

Blob Storage feature support in Azure Storage accounts

8/22/2022 • 4 minutes to read • [Edit Online](#)

Feature support is impacted by the type of account that you create and the settings that enable on that account. You can use the tables in this article to assess feature support based on these factors. The items that appear in these tables will change over time as support continues to expand.

How to use these tables

Each table uses the following icons to indicate support level:

ICON	DESCRIPTION
✓	Fully supported
□	Supported at the preview level
□	Not <i>yet</i> supported

This table describes the impact of **enabling** the capability and not the specific use of that capability. For example, if you enable the Network File System (NFS) 3.0 protocol but never use the NFS 3.0 protocol to upload a blob, a check mark in the **NFS 3.0 enabled** column indicates that feature support is not negatively impacted by merely enabling NFS 3.0 support.

Even though a feature is not be negatively impacted, it might not be compatible when used with a specific capability. For example, enabling NFS 3.0 has no impact on Azure Active Directory (Azure AD) authorization. However, you can't use Azure AD to authorize an NFS 3.0 request. See any of these articles for information about known limitations:

- [Known issues: Hierarchical namespace capability](#)
- [Known issues: Network File System \(NFS\) 3.0 protocol](#)
- [Known issues: SSH File Transfer Protocol \(SFTP\)](#)

Standard general-purpose v2 accounts

The following table describes whether a feature is supported in a standard general-purpose v2 account when you enable a hierarchical namespace (HNS), NFS 3.0 protocol, or SFTP.

IMPORTANT

This table describes the impact of **enabling** HNS, NFS, or SFTP and not the specific use of those capabilities.

STORAGE FEATURE	DEFAULT	HNS	NFS	SFTP
Access tier - archive	✓	✓	✓	✓

Storage Feature	Default	HNS	NFS	SFTP
Access tier - cool	✓	✓	✓	✓
Access tier - hot	✓	✓	✓	✓
Anonymous public access	✓	✓	✓	✓
Azure Active Directory security	✓	✓	✓ ¹	✓ ¹
Blob inventory	✓	□	□	□
Blob index tags	✓	□	□	□
Blob snapshots	✓	□	□	□
Blob Storage APIs	✓	✓	✓	✓
Blob Storage Azure CLI commands	✓	✓	✓	✓
Blob Storage events	✓	✓	□	✓
Blob Storage PowerShell commands	✓	✓	✓	✓
Blob versioning	✓	□	□	□
Blobfuse	✓	✓	✓	✓
Change feed	✓	□	□	□
Custom domains	✓	□	□	□
Customer-managed account failover	✓	□	□	□
Customer-managed keys (encryption)	✓	✓	✓	✓
Customer-provided keys (encryption)	✓	□	□	□
Data redundancy options	✓	✓	✓ ²	✓
Encryption scopes	✓	□	□	□
Immutable storage	✓	□	□	□

Storage feature	Default	HNS	NFS	SFTP
Last access time tracking for lifecycle management	✓	✓	□	✓
Lifecycle management policies (delete blob)	✓	✓	✓	✓
Lifecycle management policies (tiering)	✓	✓	✓	✓
Logging in Azure Monitor	✓	✓	□	□
Metrics in Azure Monitor	✓	✓	✓	✓
Object replication for block blobs	✓	□	□	□
Point-in-time restore for block blobs	✓	□	□	□
Soft delete for blobs	✓	✓	✓	✓
Soft delete for containers	✓	✓	✓	✓
Static websites	✓	✓	□	✓
Storage Analytics logs (classic)	✓	✓	□	✓
Storage Analytics metrics (classic)	✓	✓	✓	✓

¹ Requests that clients make by using NFS 3.0 or SFTP can't be authorized by using Azure Active Directory (AD) security.

² Only locally redundant storage (LRS) and zone-redundant storage (ZRS) are supported.

Premium block blob accounts

The following table describes whether a feature is supported in a premium block blob account when you enable a hierarchical namespace (HNS), NFS 3.0 protocol, or SFTP.

IMPORTANT

This table describes the impact of enabling HNS, NFS, or SFTP and not the specific use of those capabilities.

Storage feature	Default	HNS	NFS	SFTP
Access tier - archive	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Access tier - cool	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Access tier - hot	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Anonymous public access	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Azure Active Directory security	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> ¹	<input checked="" type="checkbox"/> ¹
Blob inventory	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Blob index tags	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Blob snapshots	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Blob Storage APIs	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Blob Storage Azure CLI commands	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Blob Storage events	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Blob Storage PowerShell commands	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Blob versioning	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Blobfuse	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Change feed	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Custom domains	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Customer-managed account failover	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Customer-managed keys (encryption)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Customer-provided keys (encryption)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Data redundancy options	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> ²	<input checked="" type="checkbox"/>
Encryption scopes	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Immutable storage	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Storage feature	Default	HNS	NFS	SFTP
Last access time tracking for lifecycle management	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Lifecycle management policies (delete blob)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Lifecycle management policies (tiering)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Logging in Azure Monitor	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Metrics in Azure Monitor	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Object replication for block blobs	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Point-in-time restore for block blobs	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Soft delete for blobs	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Soft delete for containers	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Static websites	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Storage Analytics logs (classic)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Storage Analytics metrics (classic)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

¹ Requests that clients make by using NFS 3.0 or SFTP can't be authorized by using Azure Active Directory (AD) security.

² Only locally redundant storage (LRS) and zone-redundant storage (ZRS) are supported.

See also

- [Known issues with Azure Data Lake Storage Gen2](#)
- [Known issues with Network File System \(NFS\) 3.0 protocol support in Azure Blob Storage](#)
- [Known issues with SSH File Transfer Protocol \(SFTP\) support in Azure Blob Storage \(preview\)](#)

Introduction to Azure Blob storage

8/22/2022 • 5 minutes to read • [Edit Online](#)

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 doesn't adhere to a particular data model or definition, such as text or binary data.

About Blob storage

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](#)
- [PHP](#)
- [Ruby](#)

About Azure Data Lake Storage Gen2

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

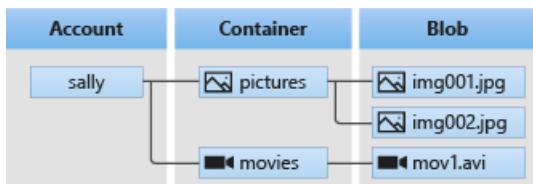
For more information about Data Lake Storage Gen2, see [Introduction to Azure Data Lake Storage Gen2](#).

Blob storage resources

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 Blob Storage 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
```

The following table describes the different types of storage accounts that are supported for Blob Storage:

TYPE OF STORAGE ACCOUNT	PERFORMANCE TIER	USAGE
General-purpose v2	Standard	Standard storage account type for blobs, file shares, queues, and tables. Recommended for most scenarios using Blob Storage or one of the other Azure Storage services.
Block blob	Premium	Premium storage account type for block blobs and append blobs. Recommended for scenarios with high transaction rates or that use smaller objects or require consistently low storage latency. Learn more about workloads for premium block blob accounts...
Page blob	Premium	Premium storage account type for page blobs only. Learn more about workloads for premium page blob accounts...

To learn more about types of storage accounts, see [Azure storage account overview](#). For information about legacy storage account types, see [Legacy storage account types](#).

To learn how to create a storage account, see [Create a storage account](#).

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, as it forms part of the unique URI used to address the container or its blobs. Follow these rules when naming a container:

- Container names can be between 3 and 63 characters long.
- Container names must start with a letter or number, and can contain only lowercase letters, numbers, and the dash (-) character.
- Two or more consecutive dash characters aren't permitted in container names.

The URI for a container is similar to:

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

For more information about naming containers, see [Naming and Referencing Containers, Blobs, and Metadata](#).

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. 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](#)

For more information about the different types of blobs, see [Understanding Block Blobs, Append Blobs, and Page Blobs](#).

The URI for a blob is similar to:

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

or

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

Follow these rules when naming a blob:

- A blob name can contain any combination of characters.
- A blob name must be at least one character long and cannot be more than 1,024 characters long, for blobs in Azure Storage.
- Blob names are case-sensitive.
- Reserved URL characters must be properly escaped.
- The number of path segments comprising the blob name cannot exceed 254. A path segment is the string between consecutive delimiter characters (*e.g.*, the forward slash '/') that corresponds to the name of a virtual directory.

NOTE

Avoid blob names that end with a dot (.), a forward slash (/), or a sequence or combination of the two. No path segments should end with a dot (.).

For more information about naming blobs, see [Naming and Referencing Containers, Blobs, and Metadata](#).

Move data to Blob storage

A number of solutions exist for migrating existing data to Blob storage:

- **AzCopy** is an easy-to-use command-line tool for Windows and Linux that copies data to and from Blob storage, across containers, or across storage accounts. For more information about AzCopy, see [Transfer data with the AzCopy v10](#).
- The **Azure Storage Data Movement library** is a .NET library for moving data between Azure Storage services. The AzCopy utility is built with the Data Movement library. For more information, see the [reference documentation](#) for the Data Movement library.

- **Azure Data Factory** supports copying data to and from Blob storage by using the account key, a shared access signature, a service principal, or managed identities for Azure resources. For more information, see [Copy data to or from Azure Blob storage by using Azure Data Factory](#).
- **Blobfuse** is a virtual file system driver for Azure Blob storage. You can use blobfuse to access your existing block blob data in your Storage account through the Linux file system. For more information, see [How to mount Blob storage as a file system with blobfuse](#).
- **Azure Data Box** service is available to transfer on-premises data to Blob storage when large datasets or network constraints make uploading data over the wire unrealistic. Depending on your data size, you can request [Azure Data Box Disk](#), [Azure Data Box](#), or [Azure Data Box Heavy](#) devices from Microsoft. You can then copy your data to those devices and ship them back to Microsoft to be uploaded into Blob storage.
- The **Azure Import/Export service** provides a way to import or export large amounts of data to and from your storage account using hard drives that you provide. For more information, see [Use the Microsoft Azure Import/Export service to transfer data to Blob storage](#).

Next steps

- [Create a storage account](#)
- [Scalability and performance targets for Blob storage](#)

Quickstart: Upload, download, and list blobs with the Azure portal

8/22/2022 • 3 minutes to read • [Edit Online](#)

In this quickstart, you learn how to use the [Azure portal](#) to create a container in Azure Storage, and to upload and download block blobs in that container.

Prerequisites

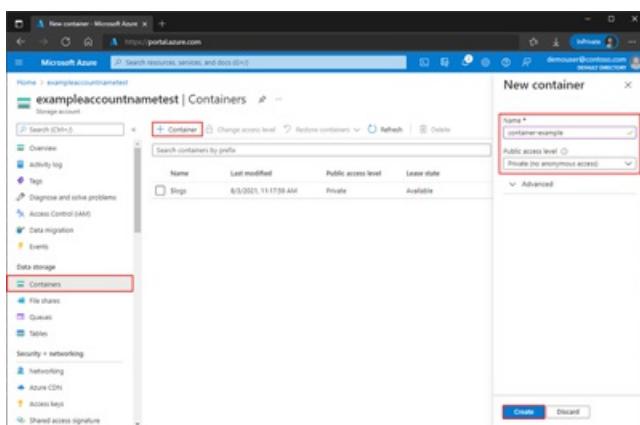
To access Azure Storage, you'll need an Azure subscription. If you don't already have a subscription, create a [free account](#) before you begin.

All access to Azure Storage takes place through a storage account. For this quickstart, create a storage account using the [Azure portal](#), Azure PowerShell, or Azure CLI. For help creating a storage account, see [Create a storage account](#).

Create a container

To create a container in the Azure portal, follow these steps:

1. Navigate to your new storage account in the Azure portal.
2. In the left menu for the storage account, scroll to the **Data storage** section, then select **Containers**.
3. Select the **+ Container** button.
4. Type a name for your new container. The container name must be lowercase, must start with a letter or number, and can include only letters, numbers, and the dash (-) character. For more information about container and blob names, see [Naming and referencing containers, blobs, and metadata](#).
5. Set the level of public access to the container. The default level is **Private (no anonymous access)**.
6. Select **Create** to create the container.



Upload a block blob

Block blobs consist of blocks of data assembled to make a blob. Most scenarios using Blob storage employ block blobs. Block blobs are ideal for storing text and binary data in the cloud, like files, images, and videos. This quickstart shows how to work with block blobs.

To upload a block blob to your new container in the Azure portal, follow these steps:

1. In the Azure portal, navigate to the container you created in the previous section.
2. Select the container to show a list of blobs it contains. This container is new, so it won't yet contain any blobs.
3. Select the **Upload** button to open the upload blade and browse your local file system to find a file to upload as a block blob. You can optionally expand the **Advanced** section to configure other settings for the upload operation.

Upload blob X

sample-container/

Files (1)

Upload

Overwrite if files already exist

^ Advanced

Authentication type (1)

Azure AD user account Account key

Blob type (1)

▼

Upload .vhdx files as page blobs (recommended)

Block size (1)

▼

Access tier (1)

▼

Upload to folder

Blob index tags (1)

Key	Value
<input type="text"/>	<input type="text"/>

Encryption scope

Use existing default container scope

Choose an existing scope

Retention policy

No retention

Choose custom retention period:

10/25/2021 ▼ 1:35:12 PM ▼

i Enable version-level immutability on the container to set a retention policy.

Upload

4. Select the **Upload** button to upload the blob.
5. Upload as many blobs as you like in this way. You'll see that the new blobs are now listed within the

container.

Download a block blob

You can download a block blob to display in the browser or save to your local file system. To download a block blob, follow these steps:

1. Navigate to the list of blobs that you uploaded in the previous section.
2. Right-click the blob you want to download, and select **Download**.

The screenshot shows the Azure Storage Explorer interface. On the left, there's a sidebar with options like 'Upload', 'Change access level', 'Refresh', 'Delete', 'Change tier', and 'Acquire lease'. Below that is a search bar labeled 'Search blobs by prefix (case-sensitive)' and a 'Add filter' button. The main area displays a table of blobs with columns: Name, Modified, Access tier, Blob type, and Size. One row is selected, showing 'blobs.png' with a modified date of 8/10/2020, 3:02:16 PM, Hot (Inferred) access tier, Block blob type, and 25.81 KiB size. To the right of the table is a context menu with various options: View/edit, Download (highlighted with a red box), Properties, Edit metadata, Generate SAS, View previous versions, View snapshots, Create snapshot, Change tier, Acquire lease, Break lease, and Delete. The 'Download' option is the first item in the list.

Delete a block blob

To delete one or more blobs in the Azure portal, follow these steps:

1. In the Azure portal, navigate to the container.
2. Display the list of blobs in the container.
3. Use the checkbox to select one or more blobs from the list.
4. Select the **Delete** button to delete the selected blobs.
5. In the dialog, confirm the deletion, and indicate whether you also want to delete blob snapshots.

The screenshot shows the Azure Storage Explorer interface. On the left, there's a sidebar with options like 'Search (Ctrl+)', 'Upload', 'Change access level', 'Refresh', 'Delete', 'Change tier', 'Acquire lease', 'Break lease', and '...'. Below that is a search bar labeled 'Search blobs by prefix (case-sensitive)' and a 'Show deleted blobs' toggle switch. The main area displays a table of blobs with columns: Name, Modified, Access tier, Blob type, and Size. Several blobs are selected for deletion, indicated by checked checkboxes in the first column. The blobs listed are: 'level1' (unchecked), 'blob1.txt' (checked), 'blob2.txt' (checked), 'blob3.txt' (checked), 'blob4.txt' (unchecked), 'blob5.txt' (unchecked), 'blob6.txt' (unchecked), 'blob7.txt' (unchecked), 'blob8.txt' (unchecked), 'blob9.txt' (unchecked), and 'logfile.txt' (unchecked). The 'Delete' button is located at the top of the blob list, just below the toolbar.

Clean up resources

To remove all the resources you created in this quickstart, you can simply delete the container. All blobs in the container will also be deleted.

To delete the container:

1. In the Azure portal, navigate to the list of containers in your storage account.
2. Select the container to delete.
3. Select the **More** button (...), and select **Delete**.
4. Confirm that you want to delete the container.

Next steps

In this quickstart, you learned how to create a container and upload a blob with Azure portal. To learn about working with Blob storage from a web app, continue to a tutorial that shows how to upload images to a storage account.

[Tutorial: Upload image data in the cloud with Azure Storage](#)

Quickstart: Use Azure Storage Explorer to create a blob

8/22/2022 • 4 minutes to read • [Edit Online](#)

In this quickstart, you learn how to use [Azure Storage Explorer](#) to create a container and a blob. Next, you learn how to download the blob to your local computer, and how to view all of the blobs in a container. You also learn how to create a snapshot of a blob, manage container access policies, and create a shared access signature.

Prerequisites

To access Azure Storage, you'll need an Azure subscription. If you don't already have a subscription, create a [free account](#) before you begin.

All access to Azure Storage takes place through a storage account. For this quickstart, create a storage account using the [Azure portal](#), Azure PowerShell, or Azure CLI. For help creating a storage account, see [Create a storage account](#).

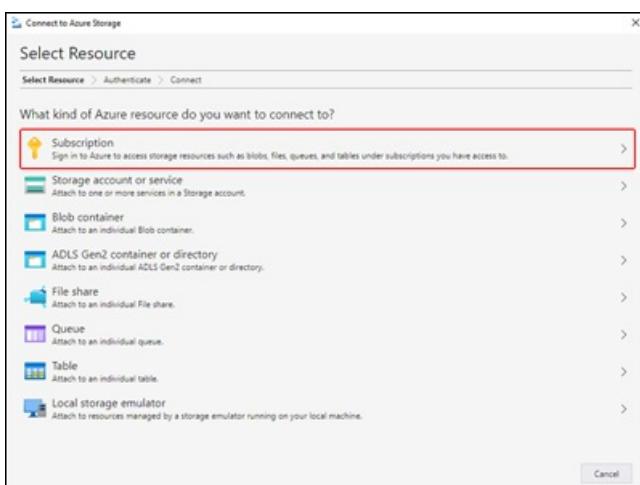
This quickstart requires that you install Azure Storage Explorer. To install Azure Storage Explorer for Windows, Macintosh, or Linux, see [Azure Storage Explorer](#).

Log in to Storage Explorer

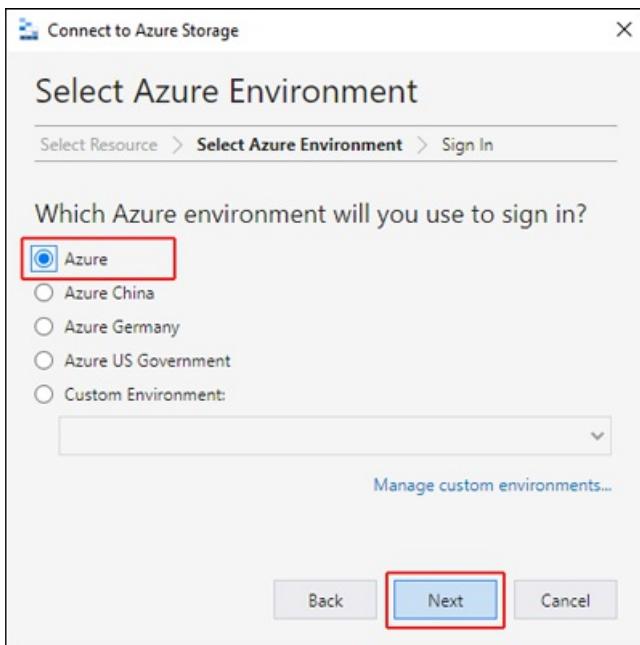
On first launch, the **Microsoft Azure Storage Explorer - Connect to Azure Storage** dialog is shown. Several resource options are displayed to which you can connect:

- Subscription
- Storage account
- Blob container
- ADLS Gen2 container or directory
- File share
- Queue
- Table
- Local storage emulator

In the **Select Resource** panel, select **Subscription**.

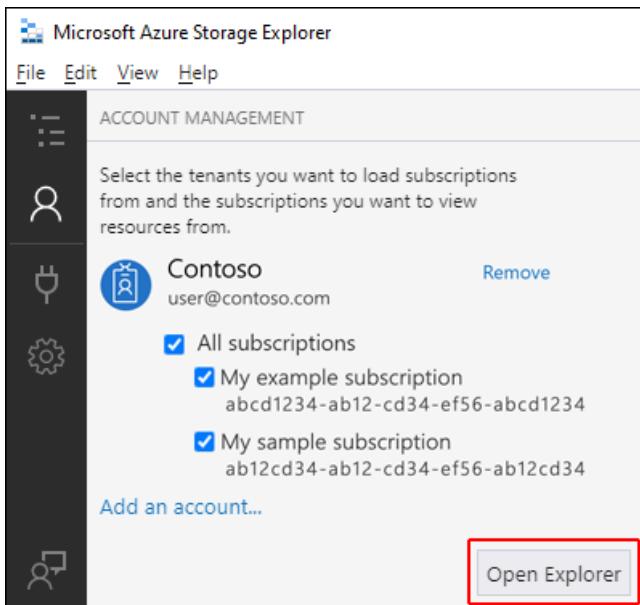


In the **Select Azure Environment** panel, select an Azure environment to sign in to. You can sign in to global Azure, a national cloud or an Azure Stack instance. Then select **Next**.

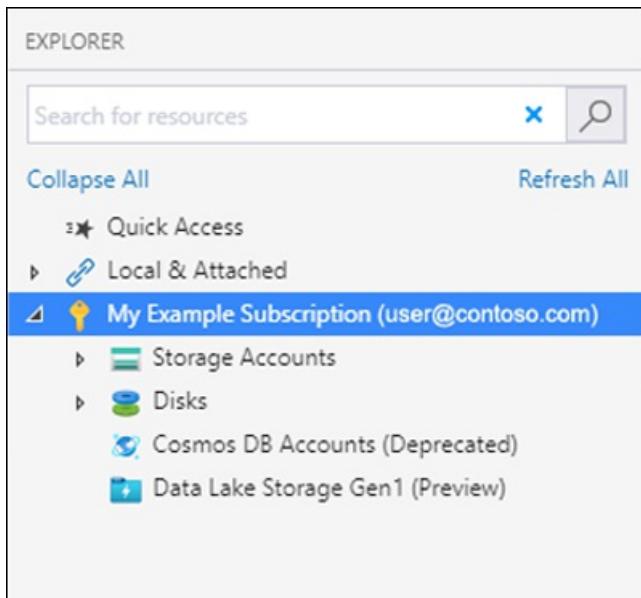


Storage Explorer will open a webpage for you to sign in.

After you successfully sign in with an Azure account, the account and the Azure subscriptions associated with that account appear under **ACCOUNT MANAGEMENT**. Select the Azure subscriptions that you want to work with, and then select **Open Explorer**.

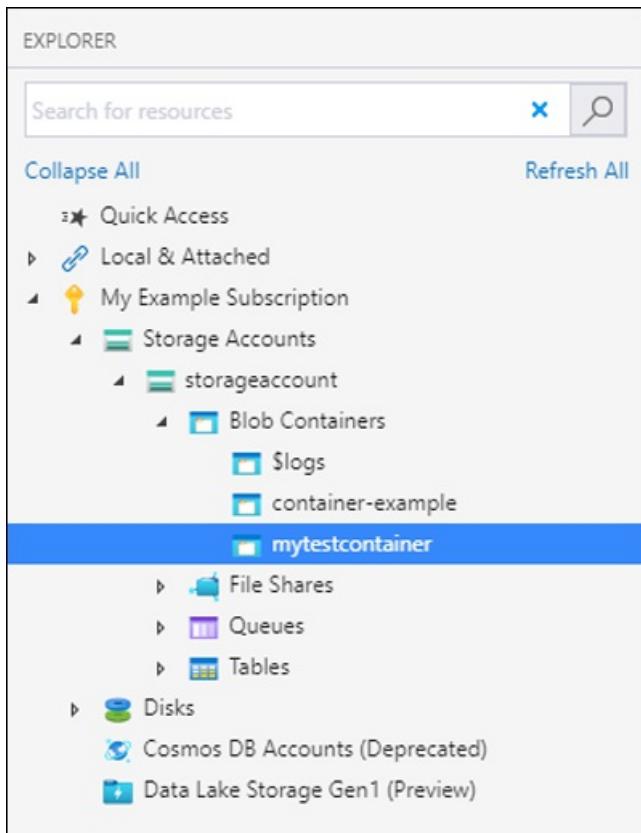


After Storage Explorer finishes connecting, it displays the **Explorer** tab. This view gives you insight to all of your Azure storage accounts as well as local storage configured through the [Azurite storage emulator](#) or [Azure Stack](#) environments.



Create a container

To create a container, expand the storage account you created in the proceeding step. Select **Blob Containers**, right-click and select **Create Blob Container**. Enter the name for your blob container. See the [Create a container](#) section for a list of rules and restrictions on naming blob containers. When complete, press **Enter** to create the blob container. Once the blob container has been successfully created, it is displayed under the **Blob Containers** folder for the selected storage account.



Upload blobs to the container

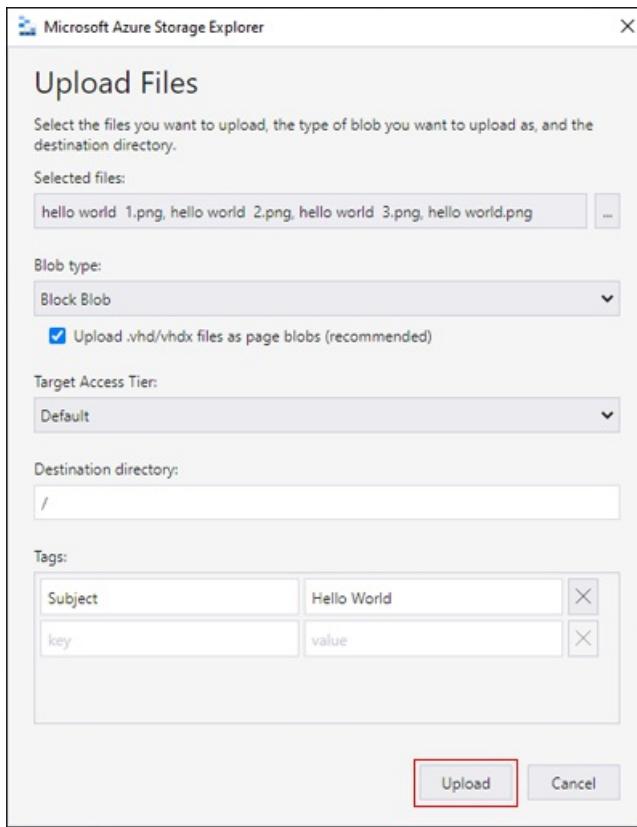
Blob storage supports block blobs, append blobs, and page blobs. VHD files used to back IaaS VMs are page blobs. Append blobs are used for logging, such as when you want to write to a file and then keep adding more information. Most files stored in Blob storage are block blobs.

On the container ribbon, select **Upload**. This operation gives you the option to upload a folder or a file.

Choose the files or folder to upload. Select the **blob type**. Acceptable choices are **Append**, **Page**, or **Block blob**.

If uploading a .vhdx file, choose **Upload .vhdx files as page blobs (recommended)**.

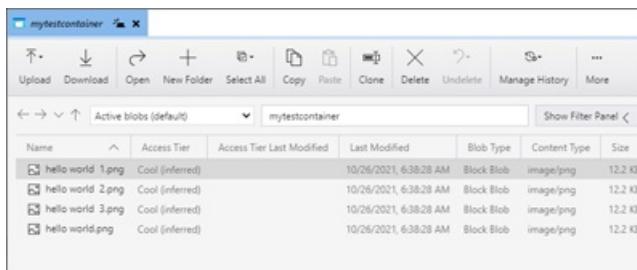
In the **Upload to folder (optional)** field either a folder name to store the files or folders in a folder under the container. If no folder is chosen, the files are uploaded directly under the container.



When you select **Upload**, the files selected are queued to upload, each file is uploaded. When the upload is complete, the results are shown in the **Activities** window.

View blobs in a container

In the **Azure Storage Explorer** application, select a container under a storage account. The main pane shows a list of the blobs in the selected container.



Download blobs

To download blobs using **Azure Storage Explorer**, with a blob selected, select **Download** from the ribbon. A file dialog opens and provides you the ability to enter a file name. Select **Save** to start the download of a blob to the local location.

Name	Access Tier	Access Tier Last Modified	Last Modified	Blob Type	Content Type
hello world 1.png	Cool (inferred)	10/26/2021, 3:11:31 PM	10/26/2021, 3:11:31 PM	Block Blob	image/png
hello world 2.png	Cool (inferred)	10/26/2021, 3:11:31 PM	10/26/2021, 3:11:31 PM	Block Blob	image/png
hello world 3.png	Cool (inferred)	10/26/2021, 3:11:31 PM	10/26/2021, 3:11:31 PM	Block Blob	image/png
hello world.png	Cool (inferred)	10/26/2021, 3:11:31 PM	10/26/2021, 3:11:31 PM	Block Blob	image/png

Manage snapshots

Azure Storage Explorer provides the capability to take and manage **snapshots** of your blobs. To take a snapshot of a blob, right-click the blob and select **Create Snapshot**. To view snapshots for a blob, right-click the blob and select **Manage history** and **Manage Snapshots**. A list of the snapshots for the blob are shown in the current tab.

Name	Access Tier	Access Tier Last Modified	Last Modified
hello world.png (2021-10-27T15:47:47.0844853Z)	Cool (inferred)	10/26/2021, 3:11:31 PM	10/26/2021, 3:11:31 PM
hello world.png (2021-10-27T16:34:46.2486188Z)	Cool (inferred)	10/26/2021, 3:11:31 PM	10/26/2021, 3:11:31 PM
hello world.png	Cool (inferred)		10/26/2021, 3:11:31 PM

Generate a shared access signature

You can use Storage Explorer to generate a shared access signatures (SAS). Right-click a storage account, container, or blob and choose **Get Shared Access Signature....** Choose the start and expiry time, and permissions for the SAS URL and select **Create**. Storage Explorer generates the SAS token with the parameters you specified and displays it for copying.

Shared Access Signature

Blob:
HelloWorld.jpg

URL:
https://hr67zeymbbyjksawinvm.blob.core.windows.net/mytestconta...

Query string:
?st=2017-11-21T00%3A44%3A00Z&se=2017-11-22T00%3A44%3A...

Back Close

When you create a SAS for a storage account, Storage Explorer generates an account SAS. For more information about the account SAS, see [Create an account SAS](#).

When you create a SAS for a container or blob, Storage Explorer generates a service SAS. For more information about the service SAS, see [Create a service SAS](#).

NOTE

When you create a SAS with Storage Explorer, the SAS is always assigned with the storage account key. Storage Explorer does not currently support creating a user delegation SAS, which is a SAS that is signed with Azure AD credentials.

Next steps

In this quickstart, you learned how to transfer files between a local disk and Azure Blob storage using **Azure Storage Explorer**. To learn more about working with Blob storage, continue to the Blob storage overview.

[Introduction to Azure Blob Storage](#)

Quickstart: Upload, download, and list blobs with PowerShell

8/22/2022 • 5 minutes to read • [Edit Online](#)

Use the Azure PowerShell module to create and manage Azure resources. You can create or manage Azure resources from the PowerShell command line or in scripts. This guide describes using PowerShell to transfer files between local disk and Azure Blob storage.

Prerequisites

To access Azure Storage, you'll need an Azure subscription. If you don't already have a subscription, then create a [free account](#) before you begin.

You will also need the Storage Blob Data Contributor role to read, write, and delete Azure Storage containers and blobs.

NOTE

This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

This quickstart requires the Azure PowerShell module Az version 0.7 or later. Run

```
Get-InstalledModule -Name Az -AllVersions | select Name,Version
```

 to find the version. If you need to install or upgrade, see [Install Azure PowerShell module](#).

Sign in to Azure

Sign in to your Azure subscription with the `Connect-AzAccount` command and follow the on-screen directions.

```
Connect-AzAccount
```

If you don't know which location you want to use, you can list the available locations. Display the list of locations by using the following code example and find the one you want to use. This example uses `eastus`. Store the location in a variable and use the variable so you can change it in one place.

```
Get-AzLocation | Select-Object -Property Location  
$Location = 'eastus'
```

Create a resource group

Create an Azure resource group with [New-AzResourceGroup](#). A resource group is a logical container into which Azure resources are deployed and managed.

```
$ResourceGroup = 'MyResourceGroup'  
New-AzResourceGroup -Name $ResourceGroup -Location $Location
```

Create a storage account

Create a standard, general-purpose storage account with LRS replication by using [New-AzStorageAccount](#). Next, get the storage account context that defines the storage account you want to use. When acting on a storage account, reference the context instead of repeatedly passing in the credentials. Use the following example to create a storage account called *mystorageaccount* with locally redundant storage (LRS) and blob encryption (enabled by default).

```
$StorageHT = @{
    ResourceGroupName = $ResourceGroup
    Name              = 'mystorageaccount'
    SkuName           = 'Standard_LRS'
    Location          = $Location
}
$StorageAccount = New-AzStorageAccount @StorageHT
$Context = $StorageAccount.Context
```

Create a container

Blobs are always uploaded into a container. You can organize groups of blobs like the way you organize your files on your computer in folders.

Set the container name, then create the container by using [New-AzStorageContainer](#). Set the permissions to `blob` to allow public access of the files. The container name in this example is *quickstartblobs*.

```
$ContainerName = 'quickstartblobs'
New-AzStorageContainer -Name $ContainerName -Context $Context -Permission Blob
```

Upload blobs to the container

Blob storage supports block blobs, append blobs, and page blobs. VHD files that back IaaS VMs are page blobs. Use append blobs for logging, such as when you want to write to a file and then keep adding more information. Most files stored in Blob storage are block blobs.

To upload a file to a block blob, get a container reference, then get a reference to the block blob in that container. Once you have the blob reference, you can upload data to it by using [Set-AzStorageBlobContent](#). This operation creates the blob if it doesn't exist, or overwrites the blob if it exists.

The following examples upload *Image001.jpg* and *Image002.png* from the *D:\Images* folder on the local disk to the container you created.

```

# upload a file to the default account (inferred) access tier
$Blob1HT = @{
    File          = 'D:\Images\Image001.jpg'
    Container     = $ContainerName
    Blob          = "Image001.jpg"
    Context       = $Context
    StandardBlobTier = 'Hot'
}
Set-AzStorageBlobContent @Blob1HT

# upload another file to the Cool access tier
$Blob2HT = @{
    File          = 'D:\Images\Image002.jpg'
    Container     = $ContainerName
    Blob          = 'Image002.png'
    Context       = $Context
    StandardBlobTier = 'Cool'
}
Set-AzStorageBlobContent @Blob2HT

# upload a file to a folder to the Archive access tier
$Blob3HT = @{
    File          = 'D:\Images\FolderName\Image003.jpg'
    Container     = $ContainerName
    Blob          = 'FolderName/Image003.jpg'
    Context       = $Context
    StandardBlobTier = 'Archive'
}
Set-AzStorageBlobContent @Blob3HT

```

Upload as many files as you like before continuing.

List the blobs in a container

Get a list of blobs in the container by using [Get-AzStorageBlob](#). This example shows just the names of the blobs uploaded.

```

Get-AzStorageBlob -Container $ContainerName -Context $Context |
    Select-Object -Property Name

```

Download blobs

Download the blobs to your local disk. For each blob you want to download, set the name and call [Get-AzStorageBlobContent](#) to download the blob.

This example downloads the blobs to *D:\Images\Downloads* on the local disk.

```

# Download first blob
$DLBlob1HT = @{
    Blob      = 'Image001.jpg'
    Container = $ContainerName
    Destination = 'D:\Images\Downloads\' 
    Context    = $Context
}
Get-AzStorageBlobContent @DLBlob1HT

# Download another blob
$DLBlob2HT = @{
    Blob      = 'Image002.png'
    Container = $ContainerName
    Destination = 'D:\Images\Downloads\' 
    Context    = $Context
}
Get-AzStorageBlobContent @DLBlob2HT

```

Data transfer with AzCopy

The AzCopy command-line utility offers high-performance, scriptable data transfer for Azure Storage. You can use AzCopy to transfer data to and from Blob storage and Azure Files. For more information about AzCopy v10, the latest version of AzCopy, see [Get started with AzCopy](#). To learn about using AzCopy v10 with Blob storage, see [Transfer data with AzCopy and Blob storage](#).

The following example uses AzCopy to upload a local file to a blob. Remember to replace the sample values with your own values:

```

azcopy login
azcopy copy 'D:\Images\Image001.jpg'
"https://$StorageAccountName.blob.core.windows.net/$ContainerName/NewGraphic.jpg"

```

Clean up resources

Remove all of the assets you've created. The easiest way to remove the assets is to delete the resource group. Removing the resource group also deletes all resources included within the group. In the following example, removing the resource group removes the storage account and the resource group itself.

```
Remove-AzResourceGroup -Name $ResourceGroup
```

Next steps

In this quickstart, you transferred files between a local file system and Azure Blob storage. To learn more about working with Blob storage by using PowerShell, select an option below.

[Manage block blobs with PowerShell](#)

[Azure PowerShell samples for Azure Blob storage](#)

Microsoft Azure PowerShell Storage cmdlets reference

- [Storage PowerShell cmdlets](#)

Microsoft Azure Storage Explorer

- [Microsoft Azure Storage Explorer](#) is a free, standalone app from Microsoft that enables you to work visually with Azure Storage data on Windows, macOS, and Linux.

Quickstart: Create, download, and list blobs with Azure CLI

8/22/2022 • 6 minutes to read • [Edit Online](#)

The Azure CLI is Azure's command-line experience for managing Azure resources. You can use it in your browser with Azure Cloud Shell. You can also install it on macOS, Linux, or Windows and run it from the command line. In this quickstart, you learn to use the Azure CLI to upload and download data to and from Azure Blob storage.

Prerequisites

To access Azure Storage, you'll need an Azure subscription. If you don't already have a subscription, create a [free account](#) before you begin.

All access to Azure Storage takes place through a storage account. For this quickstart, create a storage account using the [Azure portal](#), Azure PowerShell, or Azure CLI. For help creating a storage account, see [Create a storage account](#).

Prepare your environment for the Azure CLI

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Azure Cloud Shell Quickstart - Bash](#).
[Launch Cloud Shell](#)
- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see [How to run the Azure CLI in a Docker container](#).
 - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see [Sign in with the Azure CLI](#).
 - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
 - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).
- This article requires version 2.0.46 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

Authorize access to Blob storage

You can authorize access to Blob storage from the Azure CLI either with Azure AD credentials or by using the storage account access key. Using Azure AD credentials is recommended. This article shows how to authorize Blob storage operations using Azure AD.

Azure CLI commands for data operations against Blob storage support the `--auth-mode` parameter, which enables you to specify how to authorize a given operation. Set the `--auth-mode` parameter to `login` to authorize with Azure AD credentials. For more information, see [Authorize access to blob or queue data with Azure CLI](#).

Only Blob storage data operations support the `--auth-mode` parameter. Management operations, such as

creating a resource group or storage account, automatically use Azure AD credentials for authorization.

To begin, sign-in to your Azure account with the [az login](#).

```
az login \
--name <resource-group> \
--location <location>
```

Create a resource group

Create an Azure resource group with the [az group create](#) command. A resource group is a logical container into which Azure resources are deployed and managed.

Remember to replace placeholder values in angle brackets with your own values:

```
az group create \
--name <resource-group> \
--location <location>
```

Create a storage account

Create a general-purpose storage account with the [az storage account create](#) command. The general-purpose storage account can be used for all four services: blobs, files, tables, and queues.

Remember to replace placeholder values in angle brackets with your own values:

```
az storage account create \
--name <storage-account> \
--resource-group <resource-group> \
--location <location> \
--sku Standard_ZRS \
--encryption-services blob
```

Create a container

Blobs are always uploaded into a container. You can organize groups of blobs in containers similar to the way you organize your files on your computer in folders. Create a container for storing blobs with the [az storage container create](#) command.

The following example uses your Azure AD account to authorize the operation to create the container. Before you create the container, assign the [Storage Blob Data Contributor](#) role to yourself. Even if you are the account owner, you need explicit permissions to perform data operations against the storage account. For more information about assigning Azure roles, see [Assign an Azure role for access to blob data](#).

Remember to replace placeholder values in angle brackets with your own values:

```
az ad signed-in-user show --query objectId -o tsv | az role assignment create \
--role "Storage Blob Data Contributor" \
--assignee @- \
--scope "/subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>"\n\naz storage container create \
--account-name <storage-account> \
--name <container> \
--auth-mode login
```

IMPORTANT

Azure role assignments may take a few minutes to propagate.

You can also use the storage account key to authorize the operation to create the container. For more information about authorizing data operations with Azure CLI, see [Authorize access to blob or queue data with Azure CLI](#).

Upload a blob

Blob storage supports block blobs, append blobs, and page blobs. The examples in this quickstart show how to work with block blobs.

First, create a file to upload to a block blob. If you're using Azure Cloud Shell, use the following command to create a file:

```
vi helloworld
```

When the file opens, press **insert**. Type *Hello world*, then press **Esc**. Next, type **:x**, then press **Enter**.

In this example, you upload a blob to the container you created in the last step using the [az storage blob upload](#) command. It's not necessary to specify a file path since the file was created at the root directory. Remember to replace placeholder values in angle brackets with your own values:

```
az storage blob upload \
--account-name <storage-account> \
--container-name <container> \
--name helloworld \
--file helloworld \
--auth-mode login
```

This operation creates the blob if it doesn't already exist, and overwrites it if it does. Upload as many files as you like before continuing.

To upload multiple files at the same time, you can use the [az storage blob upload-batch](#) command.

List the blobs in a container

List the blobs in the container with the [az storage blob list](#) command. Remember to replace placeholder values in angle brackets with your own values:

```
az storage blob list \
--account-name <storage-account> \
--container-name <container> \
--output table \
--auth-mode login
```

Download a blob

Use the [az storage blob download](#) command to download the blob you uploaded earlier. Remember to replace placeholder values in angle brackets with your own values:

```
az storage blob download \
--account-name <storage-account> \
--container-name <container> \
--name helloworld \
--file ~/destination/path/for/file \
--auth-mode login
```

Data transfer with AzCopy

The AzCopy command-line utility offers high-performance, scriptable data transfer for Azure Storage. You can use AzCopy to transfer data to and from Blob storage and Azure Files. For more information about AzCopy v10, the latest version of AzCopy, see [Get started with AzCopy](#). To learn about using AzCopy v10 with Blob storage, see [Transfer data with AzCopy and Blob storage](#).

The following example uses AzCopy to upload a local file to a blob. Remember to replace the sample values with your own values:

```
azcopy login
azcopy copy 'C:\myDirectory\myTextFile.txt'
'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile.txt'
```

Clean up resources

If you want to delete the resources you created as part of this quickstart, including the storage account, delete the resource group by using the [az group delete](#) command. Remember to replace placeholder values in angle brackets with your own values:

```
az group delete \
--name <resource-group> \
--no-wait
```

Next steps

In this quickstart, you learned how to transfer files between a local file system and a container in Azure Blob storage. To learn more about working with Blob storage by using Azure CLI, select an option below.

[Manage block blobs with Azure CLI](#)

[Azure CLI samples for Blob storage](#)

Quickstart: Azure Blob Storage client library v12 for .NET

8/22/2022 • 7 minutes to read • [Edit Online](#)

Get started with the Azure Blob Storage client library v12 for .NET. Azure Blob Storage is Microsoft's object storage solution for the cloud. Follow steps to install the package and try out example code for basic tasks. Blob storage is optimized for storing massive amounts of unstructured data.

The examples in this quickstart show you how to use the Azure Blob Storage client library v12 for .NET to:

- [Get the connection string](#)
- [Create a container](#)
- [Upload a blob to a container](#)
- [List blobs in a container](#)
- [Download a blob](#)
- [Delete a container](#)

Additional resources:

- [API reference documentation](#)
- [Library source code](#)
- [Package \(NuGet\)](#)
- [Samples](#)

Prerequisites

- Azure subscription - [create one for free](#)
- Azure storage account - [create a storage account](#)
- Current [.NET Core SDK](#) for your operating system. Be sure to get the SDK and not the runtime.

Setting up

This section walks you through preparing a project to work with the Azure Blob Storage client library v12 for .NET.

Create the project

Create a .NET Core application named *BlobQuickstartV12*.

1. In a console window (such as cmd, PowerShell, or Bash), use the `dotnet new` command to create a new console app with the name *BlobQuickstartV12*. This command creates a simple "Hello World" C# project with a single source file: *Program.cs*.

```
dotnet new console -n BlobQuickstartV12
```

2. Switch to the newly created *BlobQuickstartV12* directory.

```
cd BlobQuickstartV12
```

3. Inside the `BlobQuickstartV12` directory, create another directory called `data`. This is where the blob data files will be created and stored.

```
mkdir data
```

Install the package

While still in the application directory, install the Azure Blob Storage client library for .NET package by using the `dotnet add package` command.

```
dotnet add package Azure.Storage.Blobs
```

Set up the app framework

From the project directory:

1. Open the `Program.cs` file in your editor.
2. Remove the `Console.WriteLine("Hello World!");` statement.
3. Add `using` directives.
4. Update the `Main` method declaration to support async.

Here's the code:

```
using Azure.Storage.Blobs;
using Azure.Storage.Blobs.Models;
using System;
using System.IO;
using System.Threading.Tasks;

namespace BlobQuickstartV12
{
    class Program
    {
        static async Task Main()
        {
        }
    }
}
```

Copy your credentials from the Azure portal

When the sample application makes a request to Azure Storage, it must be authorized. To authorize a request, add your storage account credentials to the application as a connection string. To view your storage account credentials, follow these steps:

1. Sign in to the [Azure portal](#).
2. Locate your storage account.
3. In the storage account menu pane, under **Security + networking**, select **Access keys**. Here, you can view the account access keys and the complete connection string for each key.

The screenshot shows the 'Access keys' pane of the Azure Storage account settings. The 'Access keys' section is highlighted with a red box. The 'Storage account name' dropdown is set to 'storagesamples'. A 'Copy to clipboard' icon is visible next to the connection string.

4. In the **Access keys** pane, select **Show keys**.
5. In the **key1** section, locate the **Connection string** value. Select the **Copy to clipboard** icon to copy the connection string. You'll add the connection string value to an environment variable in the next section.

The screenshot shows the 'Access keys' pane of the Azure Storage account settings. The 'key1' section is highlighted with a red box. The 'Connection string' field is highlighted with a red box and has a 'Copy to clipboard' icon.

Configure your storage connection string

After you copy the connection string, write it to a new environment variable on the local machine running the application. To set the environment variable, open a console window, and follow the instructions for your operating system. Replace <yourconnectionstring> with your actual connection string.

- [Windows](#)
- [Linux and macOS](#)

```
setx AZURE_STORAGE_CONNECTION_STRING "<yourconnectionstring>"
```

After you add the environment variable in Windows, you must start a new instance of the command window.

Restart programs

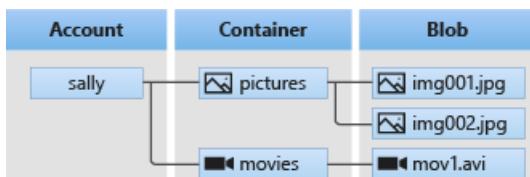
After you add the environment variable, restart any running programs that will need to read the environment variable. For example, restart your development environment or editor before you continue.

Object model

Azure 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 offers three types of resources:

- The storage account
- A container in the storage account
- A blob in the container

The following diagram shows the relationship between these resources.



Use the following .NET classes to interact with these resources:

- **BlobServiceClient**: The `BlobServiceClient` class allows you to manipulate Azure Storage resources and blob containers.
- **BlobContainerClient**: The `BlobContainerClient` class allows you to manipulate Azure Storage containers and their blobs.
- **BlobClient**: The `BlobClient` class allows you to manipulate Azure Storage blobs.

Code examples

The sample code snippets in the following sections show you how to perform basic data operations with the Azure Blob Storage client library for .NET.

Get the connection string

The code below retrieves the connection string for the storage account from the environment variable created in the [Configure your storage connection string](#) section.

Add this code inside the `Main` method:

```
Console.WriteLine("Azure Blob Storage v12 - .NET quickstart sample\n");

// Retrieve the connection string for use with the application. The storage
// connection string is stored in an environment variable on the machine
// running the application called AZURE_STORAGE_CONNECTION_STRING. If the
// environment variable is created after the application is launched in a
// console or with Visual Studio, the shell or application needs to be closed
// and reloaded to take the environment variable into account.
string connectionString = Environment.GetEnvironmentVariable("AZURE_STORAGE_CONNECTION_STRING");
```

Create a container

Decide on a name for the new container. The code below appends a GUID value to the container name to ensure that it is unique.

IMPORTANT

Container names must be lowercase. For more information about naming containers and blobs, see [Naming and Referencing Containers, Blobs, and Metadata](#).

Create an instance of the `BlobServiceClient` class. Then, call the `CreateBlobContainerAsync` method to create the container in your storage account.

Add this code to the end of the `Main` method:

```
// Create a BlobServiceClient object which will be used to create a container client
BlobServiceClient blobServiceClient = new BlobServiceClient(connectionString);

//Create a unique name for the container
string containerName = "quickstartblobs" + Guid.NewGuid().ToString();

// Create the container and return a container client object
BlobContainerClient containerClient = await blobServiceClient.CreateBlobContainerAsync(containerName);
```

Upload a blob to a container

The following code snippet:

1. Creates a text file in the local `data` directory.

- Gets a reference to a [BlobClient](#) object by calling the [GetBlobClient](#) method on the container from the [Create a container](#) section.
- Uploads the local text file to the blob by calling the [UploadAsync](#) method. This method creates the blob if it doesn't already exist, and overwrites it if it does.

Add this code to the end of the `Main` method:

```
// Create a local file in the ./data/ directory for uploading and downloading
string localPath = "./data/";
string fileName = "quickstart" + Guid.NewGuid().ToString() + ".txt";
string localFilePath = Path.Combine(localPath, fileName);

// Write text to the file
await File.WriteAllTextAsync(localFilePath, "Hello, World!");

// Get a reference to a blob
BlobClient blobClient = containerClient.GetBlobClient(fileName);

Console.WriteLine("Uploading to Blob storage as blob:\n\t {0}\n", blobClient.Uri);

// Upload data from the local file
await blobClient.UploadAsync(localFilePath, true);
```

List blobs in a container

List the blobs in the container by calling the [GetBlobsAsync](#) method. In this case, only one blob has been added to the container, so the listing operation returns just that one blob.

Add this code to the end of the `Main` method:

```
Console.WriteLine("Listing blobs...");

// List all blobs in the container
await foreach (BlobItem blobItem in containerClient.GetBlobsAsync())
{
    Console.WriteLine("\t" + blobItem.Name);
}
```

Download a blob

Download the previously created blob by calling the [DownloadToAsync](#) method. The example code adds a suffix of "DOWNLOADED" to the file name so that you can see both files in local file system.

Add this code to the end of the `Main` method:

```
// Download the blob to a local file
// Append the string "DOWNLOADED" before the .txt extension
// so you can compare the files in the data directory
string downloadFilePath = localFilePath.Replace(".txt", "DOWNLOADED.txt");

Console.WriteLine("\nDownloading blob to\n\t{0}\n", downloadFilePath);

// Download the blob's contents and save it to a file
await blobClient.DownloadToAsync(downloadFilePath);
```

Delete a container

The following code cleans up the resources the app created by deleting the entire container by using [DeleteAsync](#). It also deletes the local files created by the app.

The app pauses for user input by calling `Console.ReadLine` before it deletes the blob, container, and local files.

This is a good chance to verify that the resources were actually created correctly, before they are deleted.

Add this code to the end of the `Main` method:

```
// Clean up
Console.WriteLine("Press any key to begin clean up");
Console.ReadLine();

Console.WriteLine("Deleting blob container...");
await containerClient.DeleteAsync();

Console.WriteLine("Deleting the local source and downloaded files...");
File.Delete(localFilePath);
File.Delete(downloadFilePath);

Console.WriteLine("Done");
```

Run the code

This app creates a test file in your local `data` folder and uploads it to Blob storage. The example then lists the blobs in the container and downloads the file with a new name so that you can compare the old and new files.

Navigate to your application directory, then build and run the application.

```
dotnet build
```

```
dotnet run
```

The output of the app is similar to the following example:

```
Azure Blob Storage v12 - .NET quickstart sample

Uploading to Blob storage as blob:
    https://mystorageacct.blob.core.windows.net/quickstartblobs60c70d78-8d93-43ae-954d-
8322058cf64/quickstart2fe6c5b4-7918-46cb-96f4-8c4c5cb2fd31.txt

Listing blobs...
    quickstart2fe6c5b4-7918-46cb-96f4-8c4c5cb2fd31.txt

Downloading blob to
    ./data/quickstart2fe6c5b4-7918-46cb-96f4-8c4c5cb2fd31DOWNLOADED.txt

Press any key to begin clean up
Deleting blob container...
Deleting the local source and downloaded files...
Done
```

Before you begin the clean up process, check your `data` folder for the two files. You can open them and observe that they are identical.

After you've verified the files, press the **Enter** key to delete the test files and finish the demo.

Next steps

In this quickstart, you learned how to upload, download, and list blobs using .NET.

To see Blob storage sample apps, continue to:

Azure Blob Storage SDK v12 .NET samples

- For tutorials, samples, quick starts and other documentation, visit [Azure for .NET and .NET Core developers](#).
- To learn more about .NET Core, see [Get started with .NET in 10 minutes](#).

Quickstart: Azure Blob storage client library v11 for .NET

8/22/2022 • 10 minutes to read • [Edit Online](#)

Get started with the Azure Blob Storage client library v11 for .NET. Azure Blob Storage is Microsoft's object storage solution for the cloud. Follow steps to install the package and try out example code for basic tasks. Blob storage is optimized for storing massive amounts of unstructured data.

NOTE

This quickstart uses a legacy version of the Azure Blob storage client library. To get started with the latest version, see [Quickstart: Azure Blob storage client library v12 for .NET](#).

Use the Azure Blob Storage client library for .NET to:

- Create a container
- Set permissions on a container
- Create a blob in Azure Storage
- Download the blob to your local computer
- List all of the blobs in a container
- Delete a container

Additional resources:

- [API reference documentation](#)
- [Library source code](#)
- [Package \(NuGet\)](#)
- [Samples](#)

Prerequisites

- Azure subscription - [create one for free](#)
- Azure Storage account - [create a storage account](#)
- Current [.NET Core SDK](#) for your operating system. Be sure to get the SDK and not the runtime.

Setting up

This section walks you through preparing a project to work with the Azure Blob Storage client library for .NET.

Create the project

First, create a .NET Core application named *blob-quickstart*.

1. In a console window (such as cmd, PowerShell, or Bash), use the `dotnet new` command to create a new console app with the name *blob-quickstart*. This command creates a simple "Hello World" C# project with a single source file: *Program.cs*.

```
dotnet new console -n blob-quickstart
```

2. Switch to the newly created `blob-quickstart` folder and build the app to verify that all is well.

```
cd blob-quickstart
```

```
dotnet build
```

The expected output from the build should look something like this:

```
C:\QuickStarts\blob-quickstart> dotnet build
Microsoft (R) Build Engine version 16.0.450+ga8dc7f1d34 for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

Restore completed in 44.31 ms for C:\QuickStarts\blob-quickstart\blob-quickstart.csproj.
blob-quickstart -> C:\QuickStarts\blob-quickstart\bin\Debug\netcoreapp2.1\blob-quickstart.dll

Build succeeded.
  0 Warning(s)
  0 Error(s)

Time Elapsed 00:00:03.08
```

Install the package

While still in the application directory, install the Azure Blob Storage client library for .NET package by using the `dotnet add package` command.

```
dotnet add package Microsoft.Azure.Storage.Blob
```

Set up the app framework

From the project directory:

1. Open the `Program.cs` file in your editor
2. Remove the `Console.WriteLine` statement
3. Add `using` directives
4. Create a `ProcessAsync` method where the main code for the example will reside
5. Asynchronously call the `ProcessAsync` method from `Main`

Here's the code:

```

using System;
using System.IO;
using System.Threading.Tasks;
using Microsoft.Azure.Storage;
using Microsoft.Azure.Storage.Blob;

namespace blob_quickstart
{
    class Program
    {
        public static async Task Main()
        {
            Console.WriteLine("Azure Blob Storage - .NET quickstart sample\n");

            await ProcessAsync();

            Console.WriteLine("Press any key to exit the sample application.");
            Console.ReadLine();
        }

        private static async Task ProcessAsync()
        {
        }
    }
}

```

Copy your credentials from the Azure portal

When the sample application makes a request to Azure Storage, it must be authorized. To authorize a request, add your storage account credentials to the application as a connection string. View your storage account credentials by following these steps:

1. Navigate to the [Azure portal](#).
2. Locate your storage account.
3. In the **Settings** section of the storage account overview, select **Access keys**. Here, you can view your account access keys and the complete connection string for each key.
4. Find the **Connection string** value under **key1**, and select the **Copy** button to copy the connection string. You will add the connection string value to an environment variable in the next step.



Configure your storage connection string

After you have copied your connection string, write it to a new environment variable on the local machine running the application. To set the environment variable, open a console window, and follow the instructions for your operating system. Replace `<yourconnectionstring>` with your actual connection string.

Windows

```
setx AZURE_STORAGE_CONNECTION_STRING "<yourconnectionstring>"
```

After you add the environment variable in Windows, you must start a new instance of the command window.

Linux

```
export AZURE_STORAGE_CONNECTION_STRING=<yourconnectionstring>"
```

MacOS

```
export AZURE_STORAGE_CONNECTION_STRING=<yourconnectionstring>"
```

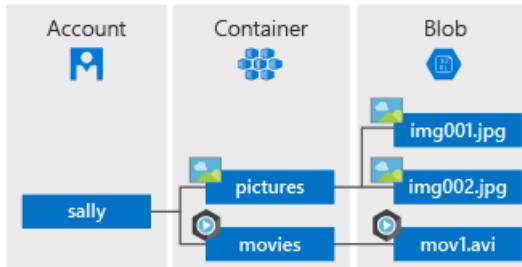
After you add the environment variable, restart any running programs that will need to read the environment variable. For example, restart your development environment or editor before continuing.

Object model

Azure 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 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.



Use the following .NET classes to interact with these resources:

- [CloudStorageAccount](#): The `CloudStorageAccount` class represents your Azure storage account. Use this class to authorize access to Blob storage using your account access keys.
- [CloudBlobClient](#): The `CloudBlobClient` class provides a point of access to the Blob service in your code.
- [CloudBlobContainer](#): The `CloudBlobContainer` class represents a blob container in your code.
- [CloudBlockBlob](#): The `CloudBlockBlob` object represents a block blob in your code. Block blobs are made up of blocks of data that can be managed individually.

Code examples

These example code snippets show you how to perform the following with the Azure Blob storage client library for .NET:

- [Authenticate the client](#)
- [Create a container](#)
- [Set permissions on a container](#)
- [Upload blobs to a container](#)
- [List the blobs in a container](#)
- [Download blobs](#)
- [Delete a container](#)

Authenticate the client

The code below checks that the environment variable contains a connection string that can be parsed to create a

`CloudStorageAccount` object pointing to the storage account. To check that the connection string is valid, use the `TryParse` method. If `TryParse` is successful, it initializes the `storageAccount` variable and returns `true`.

Add this code inside the `ProcessAsync` method:

```
// Retrieve the connection string for use with the application. The storage
// connection string is stored in an environment variable on the machine
// running the application called AZURE_STORAGE_CONNECTION_STRING. If the
// environment variable is created after the application is launched in a
// console or with Visual Studio, the shell or application needs to be closed
// and reloaded to take the environment variable into account.
string storageConnectionString = Environment.GetEnvironmentVariable("AZURE_STORAGE_CONNECTION_STRING");

// Check whether the connection string can be parsed.
CloudStorageAccount storageAccount;
if (CloudStorageAccount.TryParse(storageConnectionString, out storageAccount))
{
    // If the connection string is valid, proceed with operations against Blob
    // storage here.
    // ADD OTHER OPERATIONS HERE
}
else
{
    // Otherwise, let the user know that they need to define the environment variable.
    Console.WriteLine(
        "A connection string has not been defined in the system environment variables. " +
        "Add an environment variable named 'AZURE_STORAGE_CONNECTION_STRING' with your storage " +
        "connection string as a value.");
    Console.WriteLine("Press any key to exit the application.");
    Console.ReadLine();
}
```

NOTE

To perform the rest of the operations in this article, replace `// ADD OTHER OPERATIONS HERE` in the code above with the code snippets in the following sections.

Create a container

To create the container, first create an instance of the `CloudBlobClient` object, which points to Blob storage in your storage account. Next, create an instance of the `CloudBlobContainer` object, then create the container.

In this case, the code calls the `CreateAsync` method to create the container. A GUID value is appended to the container name to ensure that it is unique. In a production environment, it's often preferable to use the `CreateIfNotExistsAsync` method to create a container only if it does not already exist.

IMPORTANT

Container names must be lowercase. For more information about naming containers and blobs, see [Naming and Referencing Containers, Blobs, and Metadata](#).

```

// Create the CloudBlobClient that represents the
// Blob storage endpoint for the storage account.
CloudBlobClient cloudBlobClient = storageAccount.CreateCloudBlobClient();

// Create a container called 'quickstartblobs' and
// append a GUID value to it to make the name unique.
CloudBlobContainer cloudBlobContainer =
    cloudBlobClient.GetContainerReference("quickstartblobs" +
        Guid.NewGuid().ToString());
await cloudBlobContainer.CreateAsync();

```

Set permissions on a container

Set permissions on the container so that any blobs in the container are public. If a blob is public, it can be accessed anonymously by any client.

```

// Set the permissions so the blobs are public.
BlobContainerPermissions permissions = new BlobContainerPermissions
{
    PublicAccess = BlobContainerPublicAccessType.Blob
};
await cloudBlobContainer.SetPermissionsAsync(permissions);

```

Upload blobs to a container

The following code snippet gets a reference to a `CloudBlockBlob` object by calling the [GetBlockBlobReference](#) method on the container created in the previous section. It then uploads the selected local file to the blob by calling the [UploadFromFileAsync](#) method. This method creates the blob if it doesn't already exist, and overwrites it if it does.

```

// Create a file in your local MyDocuments folder to upload to a blob.
string localPath = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
string localFileName = "QuickStart_" + Guid.NewGuid().ToString() + ".txt";
string sourceFile = Path.Combine(localPath, localFileName);
// Write text to the file.
File.WriteAllText(sourceFile, "Hello, World!");

Console.WriteLine("Temp file = {0}", sourceFile);
Console.WriteLine("Uploading to Blob storage as blob '{0}'", localFileName);

// Get a reference to the blob address, then upload the file to the blob.
// Use the value of localFileName for the blob name.
CloudBlockBlob cloudBlockBlob = cloudBlobContainer.GetBlockBlobReference(localFileName);
await cloudBlockBlob.UploadFromFileAsync(sourceFile);

```

List the blobs in a container

List the blobs in the container by using the [ListBlobsSegmentedAsync](#) method. In this case, only one blob has been added to the container, so the listing operation returns just that one blob.

If there are too many blobs to return in one call (by default, more than 5000), then the [ListBlobsSegmentedAsync](#) method returns a segment of the total result set and a continuation token. To retrieve the next segment of blobs, you provide in the continuation token returned by the previous call, and so on, until the continuation token is null. A null continuation token indicates that all of the blobs have been retrieved. The code shows how to use the continuation token for the sake of best practices.

```
// List the blobs in the container.
Console.WriteLine("List blobs in container.");
BlobContinuationToken blobContinuationToken = null;
do
{
    var results = await cloudBlobContainer.ListBlobsSegmentedAsync(null, blobContinuationToken);
    // Get the value of the continuation token returned by the listing call.
    blobContinuationToken = results.ContinuationToken;
    foreach (IListBlobItem item in results.Results)
    {
        Console.WriteLine(item.Uri);
    }
} while (blobContinuationToken != null); // Loop while the continuation token is not null.
```

Download blobs

Download the blob created previously to your local file system by using the [DownloadToFileAsync](#) method. The example code adds a suffix of "_DOWNLOADED" to the blob name so that you can see both files in local file system.

```
// Download the blob to a local file, using the reference created earlier.
// Append the string "_DOWNLOADED" before the .txt extension so that you
// can see both files in MyDocuments.
string destinationFile = sourceFile.Replace(".txt", "_DOWNLOADED.txt");
Console.WriteLine("Downloading blob to {0}", destinationFile);
await cloudBlockBlob.DownloadToFileAsync(destinationFile, FileMode.Create);
```

Delete a container

The following code cleans up the resources the app created by deleting the entire container using [CloudBlobContainer.DeleteAsync](#). You can also delete the local files if you like.

```
Console.WriteLine("Press the 'Enter' key to delete the example files, " +
    "example container, and exit the application.");
Console.ReadLine();
// Clean up resources. This includes the container and the two temp files.
Console.WriteLine("Deleting the container");
if (cloudBlobContainer != null)
{
    await cloudBlobContainer.DeleteIfExistsAsync();
}
Console.WriteLine("Deleting the source, and downloaded files");
File.Delete(sourceFile);
File.Delete(destinationFile);
```

Run the code

This app creates a test file in your local *MyDocuments* folder and uploads it to Blob storage. The example then lists the blobs in the container and downloads the file with a new name so that you can compare the old and new files.

Navigate to your application directory, then build and run the application.

```
dotnet build
```

```
dotnet run
```

The output of the app is similar to the following example:

```
Azure Blob storage - .NET Quickstart example

Created container 'quickstartblobs33c90d2a-eabd-4236-958b-5cc5949e731f'

Temp file = C:\Users\myusername\Documents\QuickStart_c5e7f24f-a7f8-4926-a9da-96
97c748f4db.txt
Uploading to Blob storage as blob 'QuickStart_c5e7f24f-a7f8-4926-a9da-9697c748f
4db.txt'

Listing blobs in container.
https://storagesamples.blob.core.windows.net/quickstartblobs33c90d2a-eabd-4236-
958b-5cc5949e731f/QuickStart_c5e7f24f-a7f8-4926-a9da-9697c748f4db.txt

Downloading blob to C:\Users\myusername\Documents\QuickStart_c5e7f24f-a7f8-4926
-a9da-9697c748f4db_DOWNLOADED.txt

Press any key to delete the example files and example container.
```

When you press the **Enter** key, the application deletes the storage container and the files. Before you delete them, check your *MyDocuments* folder for the two files. You can open them and observe that they are identical. Copy the blob's URL from the console window and paste it into a browser to view the contents of the blob.

After you've verified the files, hit any key to finish the demo and delete the test files.

Next steps

In this quickstart, you learned how to upload, download, and list blobs using .NET.

To learn how to create a web app that uploads an image to Blob storage, continue to:

[Upload and process an image](#)

- To learn more about .NET Core, see [Get started with .NET in 10 minutes](#).
- To explore a sample application that you can deploy from Visual Studio for Windows, see the [.NET Photo Gallery Web Application Sample with Azure Blob Storage](#).

Quickstart: Manage blobs with Java v12 SDK

8/22/2022 • 8 minutes to read • [Edit Online](#)

In this quickstart, you learn to manage blobs by using Java. Blobs are objects that can hold large amounts of text or binary data, including images, documents, streaming media, and archive data. You'll upload, download, and list blobs, and you'll create and delete containers.

Additional resources:

- [API reference documentation](#)
- [Library source code](#)
- [Package \(Maven\)](#)
- [Samples](#)

Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- An Azure Storage account. [Create a storage account](#).
- [Java Development Kit \(JDK\)](#) version 8 or above.
- [Apache Maven](#).

Setting up

This section walks you through preparing a project to work with the Azure Blob Storage client library v12 for Java.

Create the project

Create a Java application named *blob-quickstart-v12*.

1. In a console window (such as cmd, PowerShell, or Bash), use Maven to create a new console app with the name *blob-quickstart-v12*. Type the following **mvn** command to create a "Hello world!" Java project.

- [PowerShell](#)
- [Bash](#)

```
mvn archetype:generate `--define interactiveMode=n` --define groupId=com.blobs.quickstart` --define artifactId=blob-quickstart-v12` --define archetypeArtifactId=maven-archetype-quickstart` --define archetypeVersion=1.4
```

2. The output from generating the project should look something like this:

```
[INFO] Scanning for projects...
[INFO]
[INFO] -----< org.apache.maven:standalone-pom >-----
[INFO] Building Maven Stub Project (No POM) 1
[INFO] -----[ pom ]-----
[INFO]
[INFO] >>> maven-archetype-plugin:3.1.2:generate (default-cli) > generate-sources @ standalone-pom
>>>
[INFO]
[INFO] <<< maven-archetype-plugin:3.1.2:generate (default-cli) < generate-sources @ standalone-pom
<<<
[INFO]
[INFO]
[INFO] --- maven-archetype-plugin:3.1.2:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Batch mode
[INFO] -----
[INFO] Using following parameters for creating project from Archetype: maven-archetype-quickstart:1.4
[INFO] -----
[INFO] Parameter: groupId, Value: com.blobs.quickstart
[INFO] Parameter: artifactId, Value: blob-quickstart-v12
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: com.blobs.quickstart
[INFO] Parameter: packageInPathFormat, Value: com/blobs/quickstart
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: com.blobs.quickstart
[INFO] Parameter: groupId, Value: com.blobs.quickstart
[INFO] Parameter: artifactId, Value: blob-quickstart-v12
[INFO] Project created from Archetype in dir: C:\QuickStarts\blob-quickstart-v12
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 7.056 s
[INFO] Finished at: 2019-10-23T11:09:21-07:00
[INFO] -----
```

```

3. Switch to the newly created *blob-quickstart-v12* folder.

```
cd blob-quickstart-v12
```

4. In side the *blob-quickstart-v12* directory, create another directory called *data*. This is where the blob data files will be created and stored.

```
mkdir data
```

## Install the package

Open the *pom.xml* file in your text editor. Add the following dependency element to the group of dependencies.

```
<dependency>
 <groupId>com.azure</groupId>
 <artifactId>azure-storage-blob</artifactId>
 <version>12.13.0</version>
</dependency>
```

## Set up the app framework

From the project directory:

1. Navigate to the */src/main/java/com/blobs/quickstart* directory

2. Open the `App.java` file in your editor
3. Delete the `System.out.println("Hello world!");` statement
4. Add `import` directives

Here's the code:

```
package com.blobs.quickstart;

/**
 * Azure blob storage v12 SDK quickstart
 */
import com.azure.storage.blob.*;
import com.azure.storage.blob.models.*;
import java.io.*;

public class App
{
 public static void main(String[] args) throws IOException
 {
 }
}
```

### Copy your credentials from the Azure portal

When the sample application makes a request to Azure Storage, it must be authorized. To authorize a request, add your storage account credentials to the application as a connection string. To view your storage account credentials, follow these steps:

1. Sign in to the [Azure portal](#).
2. Locate your storage account.
3. In the storage account menu pane, under **Security + networking**, select **Access keys**. Here, you can view the account access keys and the complete connection string for each key.

The screenshot shows the 'storagesamples' storage account in the Azure portal. The 'Access keys' section is selected in the left sidebar. The 'Show keys' button is highlighted with a red box. The page displays information about access keys, including a note to keep them secure and update them often. It also shows the storage account name 'storagesamples'.

4. In the Access keys pane, select Show keys.
5. In the key1 section, locate the Connection string value. Select the Copy to clipboard icon to copy the connection string. You'll add the connection string value to an environment variable in the next section.

The screenshot shows the 'Access keys' blade for 'key1'. The 'Connection string' field is highlighted with a red box, and its copy-to-clipboard icon is also highlighted with a red box.

### Configure your storage connection string

After you copy the connection string, write it to a new environment variable on the local machine running the application. To set the environment variable, open a console window, and follow the instructions for your

operating system. Replace `<yourconnectionstring>` with your actual connection string.

- [Windows](#)
- [Linux and macOS](#)

```
setx AZURE_STORAGE_CONNECTION_STRING "<yourconnectionstring>"
```

After you add the environment variable in Windows, you must start a new instance of the command window.

#### Restart programs

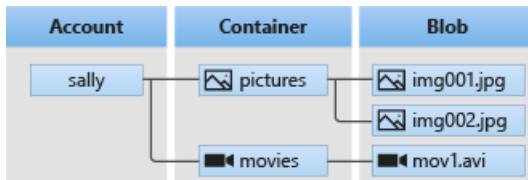
After you add the environment variable, restart any running programs that will need to read the environment variable. For example, restart your development environment or editor before you continue.

## Object model

Azure 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 offers three types of resources:

- The storage account
- A container in the storage account
- A blob in the container

The following diagram shows the relationship between these resources.



Use the following Java classes to interact with these resources:

- [BlobServiceClient](#): The `BlobServiceClient` class allows you to manipulate Azure Storage resources and blob containers. The storage account provides the top-level namespace for the Blob service.
- [BlobServiceClientBuilder](#): The `BlobServiceClientBuilder` class provides a fluent builder API to help aid the configuration and instantiation of `BlobServiceClient` objects.
- [BlobContainerClient](#): The `BlobContainerClient` class allows you to manipulate Azure Storage containers and their blobs.
- [BlobClient](#): The `BlobClient` class allows you to manipulate Azure Storage blobs.
- [BlobItem](#): The `BlobItem` class represents individual blobs returned from a call to `listBlobs`.

## Code examples

These example code snippets show you how to perform the following with the Azure Blob Storage client library for Java:

- [Get the connection string](#)
- [Create a container](#)
- [Upload blobs to a container](#)
- [List the blobs in a container](#)
- [Download blobs](#)
- [Delete a container](#)

## Get the connection string

The code below retrieves the connection string for the storage account from the environment variable created in the [Configure your storage connection string](#) section.

Add this code inside the `Main` method:

```
System.out.println("Azure Blob Storage v12 - Java quickstart sample\n");

// Retrieve the connection string for use with the application. The storage
// connection string is stored in an environment variable on the machine
// running the application called AZURE_STORAGE_CONNECTION_STRING. If the environment variable
// is created after the application is launched in a console or with
// Visual Studio, the shell or application needs to be closed and reloaded
// to take the environment variable into account.
String connectStr = System.getenv("AZURE_STORAGE_CONNECTION_STRING");
```

## Create a container

Decide on a name for the new container. The code below appends a UUID value to the container name to ensure that it is unique.

### IMPORTANT

Container names must be lowercase. For more information about naming containers and blobs, see [Naming and Referencing Containers, Blobs, and Metadata](#).

Next, create an instance of the `BlobContainerClient` class, then call the `create` method to actually create the container in your storage account.

Add this code to the end of the `Main` method:

```
// Create a BlobServiceClient object which will be used to create a container client
BlobServiceClient blobServiceClient = new
BlobServiceClientBuilder().connectionString(connectStr).buildClient();

//Create a unique name for the container
String containerName = "quickstartblobs" + java.util.UUID.randomUUID();

// Create the container and return a container client object
BlobContainerClient containerClient = blobServiceClient.createBlobContainer(containerName);
```

## Upload blobs to a container

The following code snippet:

1. Creates a text file in the local `data` directory.
2. Gets a reference to a `BlobClient` object by calling the `getBlobClient` method on the container from the [Create a container](#) section.
3. Uploads the local text file to the blob by calling the `uploadFromFile` method. This method creates the blob if it doesn't already exist, but will not overwrite it if it does.

Add this code to the end of the `Main` method:

```

// Create a local file in the ./data/ directory for uploading and downloading
String localPath = "./data/";
String fileName = "quickstart" + java.util.UUID.randomUUID() + ".txt";
File localFile = new File(localPath + fileName);

// Write text to the file
FileWriter writer = new FileWriter(localPath + fileName, true);
writer.write("Hello, World!");
writer.close();

// Get a reference to a blob
BlobClient blobClient = containerClient.getBlobClient(fileName);

System.out.println("\nUploading to Blob storage as blob:\n\t" + blobClient.getBlobUrl());

// Upload the blob
blobClient.uploadFromFile(localPath + fileName);

```

## List the blobs in a container

List the blobs in the container by calling the [listBlobs](#) method. In this case, only one blob has been added to the container, so the listing operation returns just that one blob.

Add this code to the end of the `Main` method:

```

System.out.println("\nListing blobs...");

// List the blob(s) in the container.
for (BlobItem blobItem : containerClient.listBlobs()) {
 System.out.println("\t" + blobItem.getName());
}

```

## Download blobs

Download the previously created blob by calling the [downloadToFile](#) method. The example code adds a suffix of "DOWNLOAD" to the file name so that you can see both files in local file system.

Add this code to the end of the `Main` method:

```

// Download the blob to a local file
// Append the string "DOWNLOAD" before the .txt extension so that you can see both files.
String downloadFileName = fileName.replace(".txt", "DOWNLOAD.txt");
File downloadedFile = new File(localPath + downloadFileName);

System.out.println("\nDownloading blob to\n\t" + localPath + downloadFileName);

blobClient.downloadToFile(localPath + downloadFileName);

```

## Delete a container

The following code cleans up the resources the app created by removing the entire container using the [delete](#) method. It also deletes the local files created by the app.

The app pauses for user input by calling `System.console().readLine()` before it deletes the blob, container, and local files. This is a good chance to verify that the resources were created correctly, before they are deleted.

Add this code to the end of the `Main` method:

```
// Clean up
System.out.println("\nPress the Enter key to begin clean up");
System.console().readLine();

System.out.println("Deleting blob container...");
containerClient.delete();

System.out.println("Deleting the local source and downloaded files...");
localFile.delete();
downloadedFile.delete();

System.out.println("Done");
```

## Run the code

This app creates a test file in your local folder and uploads it to Blob storage. The example then lists the blobs in the container and downloads the file with a new name so that you can compare the old and new files.

Navigate to the directory containing the *pom.xml* file and compile the project by using the following `mvn` command.

```
mvn compile
```

Then, build the package.

```
mvn package
```

Run the following `mvn` command to execute the app.

```
mvn exec:java -Dexec.mainClass="com.blobs.quickstart.App" -Dexec.cleanupDaemonThreads=false
```

The output of the app is similar to the following example:

```
Azure Blob Storage v12 - Java quickstart sample

Uploading to Blob storage as blob:
 https://mystorageacct.blob.core.windows.net/quickstartblobsf9aa68a5-260e-47e6-bea2-
2dcfcfa1fd9a/quickstarta9c3a53e-ae9d-4863-8b34-f3d807992d65.txt

Listing blobs...
 quickstarta9c3a53e-ae9d-4863-8b34-f3d807992d65.txt

Downloading blob to
 ./data/quickstarta9c3a53e-ae9d-4863-8b34-f3d807992d65DOWNLOAD.txt

Press the Enter key to begin clean up

Deleting blob container...
Deleting the local source and downloaded files...
Done
```

Before you begin the clean up process, check your *data* folder for the two files. You can open them and observe that they are identical.

After you've verified the files, press the **Enter** key to delete the test files and finish the demo.

## Next steps

In this quickstart, you learned how to upload, download, and list blobs using Java.

To see Blob storage sample apps, continue to:

[Azure Blob Storage SDK v12 Java samples](#)

- To learn more, see the [Azure SDK for Java](#).
- For tutorials, samples, quickstarts, and other documentation, visit [Azure for Java cloud developers](#).

# Quickstart: Manage blobs with Java v8 SDK

8/22/2022 • 8 minutes to read • [Edit Online](#)

In this quickstart, you learn to manage blobs by using Java. Blobs are objects that can hold large amounts of text or binary data, including images, documents, streaming media, and archive data. You'll upload, download, and list blobs. You'll also create, set permissions on, and delete containers.

## NOTE

This quickstart uses a legacy version of the Azure Blob storage client library. To get started with the latest version, see [Quickstart: Manage blobs with Java v12 SDK](#).

## Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- An Azure Storage account. [Create a storage account](#).
- An IDE that has Maven integration. This guide uses [Eclipse](#) with the "Eclipse IDE for Java Developers" configuration.

## Download the sample application

The [sample application](#) is a basic console application.

Use [git](#) to download a copy of the application to your development environment.

```
git clone https://github.com/Azure-Samples/storage-blobs-java-quickstart.git
```

This command clones the repository to your local git folder. To open the project, launch Eclipse and close the Welcome screen. Select **File** then **Open Projects from File System**. Make sure **Detect and configure project natures** is checked. Select **Directory** then navigate to where you stored the cloned repository. Inside the cloned repository, select the **blobAzureApp** folder. Make sure the **blobAzureApp** project appears as an Eclipse project, then select **Finish**.

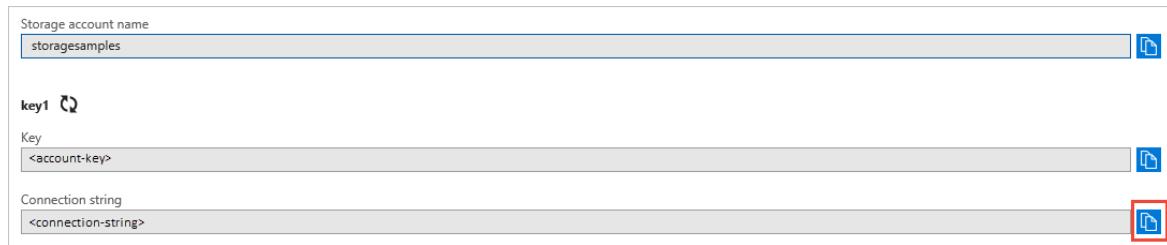
Once the project completes importing, open **AzureApp.java** (located in **blobQuickstart.blobAzureApp** inside of **src/main/java**), and replace the **accountname** and **accountkey** inside of the **storageConnectionString** string. Then run the application. Specific instructions for completing these tasks are described in the following sections.

## Copy your credentials from the Azure portal

The sample application needs to authenticate access to your storage account. To authenticate, add your storage account credentials to the application as a connection string. View your storage account credentials by following these steps:

1. Sign in to the [Azure portal](#).
2. Locate your storage account.
3. In the **Settings** section of the storage account overview, select **Access keys**. Here, you can view your account access keys and the complete connection string for each key.

4. Find the **Connection string** value under **key1**, and select the **Copy** button to copy the connection string. You will add the connection string value to an environment variable in the next step.



## Configure your storage connection string

In the application, you must provide the connection string for your storage account. Open the `AzureApp.java` file. Find the `storageConnectionString` variable and paste the connection string value that you copied in the previous section. Your `storageConnectionString` variable should look similar to the following code example:

```
public static final String storageConnectionString =
 "DefaultEndpointsProtocol=https;" +
 "AccountName=<account-name>" +
 "AccountKey=<account-key>";
```

## Run the sample

This sample application creates a test file in your default directory (`C:\Users<user>\AppData\Local\Temp`, for Windows users), uploads it to Blob storage, lists the blobs in the container, then downloads the file with a new name so you can compare the old and new files.

Run the sample using Maven at the command line. Open a shell and navigate to `blobAzureApp` inside of your cloned directory. Then enter `mvn compile exec:java`.

The following example shows the output if you were to run the application on Windows.

```
Azure Blob storage quick start sample
Creating container: quickstartcontainer
Creating a sample file at: C:\Users\<user>\AppData\Local\Temp\sampleFile514658495642546986.txt
Uploading the sample file
URI of blob is:
https://myexamplesacct.blob.core.windows.net/quickstartcontainer/sampleFile514658495642546986.txt
The program has completed successfully.
Press the 'Enter' key while in the console to delete the sample files, example container, and exit the application.

Deleting the container
Deleting the source, and downloaded files
```

Before you continue, check your default directory (`C:\Users<user>\AppData\Local\Temp`, for Windows users) for the sample file. Copy the URL for the blob out of the console window and paste it into a browser to view the contents of the file in Blob storage. If you compare the sample file in your directory with the contents stored in Blob storage, you will see that they are the same.

### NOTE

You can also use a tool such as the [Azure Storage Explorer](#) to view the files in Blob storage. Azure Storage Explorer is a free cross-platform tool that allows you to access your storage account information.

After you've verified the files, press the **Enter** key to complete the demo and delete the test files. Now that you know what the sample does, open the **AzureApp.java** file to look at the code.

## Understand the sample code

Next, we walk through the sample code so that you can understand how it works.

### Get references to the storage objects

The first thing to do is create the references to the objects used to access and manage Blob storage. These objects build on each other -- each is used by the next one in the list.

- Create an instance of the **CloudStorageAccount** object pointing to the storage account.

The **CloudStorageAccount** object is a representation of your storage account and it allows you to set and access storage account properties programmatically. Using the **CloudStorageAccount** object you can create an instance of the **CloudBlobClient**, which is necessary to access the blob service.

- Create an instance of the **CloudBlobClient** object, which points to the **Blob service** in your storage account.

The **CloudBlobClient** provides you a point of access to the blob service, allowing you to set and access Blob storage properties programmatically. Using the **CloudBlobClient** you can create an instance of the **CloudBlobContainer** object, which is necessary to create containers.

- Create an instance of the **CloudBlobContainer** object, which represents the container you are accessing. Use containers to organize your blobs like you use folders on your computer to organize your files.

Once you have the **CloudBlobContainer**, you can create an instance of the **CloudBlockBlob** object that points to the specific blob you're interested in, and perform an upload, download, copy, or other operation.

#### IMPORTANT

Container names must be lowercase. For more information about containers, see [Naming and Referencing Containers, Blobs, and Metadata](#).

### Create a container

In this section, you create an instance of the objects, create a new container, and then set permissions on the container so the blobs are public and can be accessed with just a URL. The container is called **quickstartcontainer**.

This example uses **CreateIfNotExists** because we want to create a new container each time the sample is run. In a production environment, where you use the same container throughout an application, it's better practice to only call **CreateIfNotExists** once. Alternatively, you can create the container ahead of time so you don't need to create it in the code.

```
// Parse the connection string and create a blob client to interact with Blob storage
storageAccount = CloudStorageAccount.parse(storageConnectionString);
blobClient = storageAccount.createCloudBlobClient();
container = blobClient.getContainerReference("quickstartcontainer");

// Create the container if it does not exist with public access.
System.out.println("Creating container: " + container.getName());
container.createIfNotExists(BlobContainerPublicAccessType.CONTAINER, new BlobRequestOptions(), new
OperationContext());
```

### Upload blobs to the container

To upload a file to a block blob, get a reference to the blob in the target container. Once you have the blob reference, you can upload data to it by using [CloudBlockBlob.Upload](#). This operation creates the blob if it doesn't already exist, or overwrites the blob if it already exists.

The sample code creates a local file to be used for the upload and download, storing the file to be uploaded as **source** and the name of the blob in **blob**. The following example uploads the file to your container called **quickstartcontainer**.

```
//Creating a sample file
sourceFile = File.createTempFile("sampleFile", ".txt");
System.out.println("Creating a sample file at: " + sourceFile.toString());
Writer output = new BufferedWriter(new FileWriter(sourceFile));
output.write("Hello Azure!");
output.close();

//Getting a blob reference
CloudBlockBlob blob = container.getBlockBlobReference(sourceFile.getName());

//Creating blob and uploading file to it
System.out.println("Uploading the sample file ");
blob.uploadFromFile(sourceFile.getAbsolutePath());
```

There are several `upload` methods including [upload](#), [uploadBlock](#), [uploadFullBlob](#), [uploadStandardBlobTier](#), and [uploadText](#) which you can use with Blob storage. For example, if you have a string, you can use the `UploadText` method rather than the `upload` method.

Block blobs can be any type of text or binary file. Page blobs are primarily used for the VHD files that back IaaS VMs. Use append blobs for logging, such as when you want to write to a file and then keep adding more information. Most objects stored in Blob storage are block blobs.

### List the blobs in a container

You can get a list of files in the container using [CloudBlobContainer.ListBlobs](#). The following code retrieves the list of blobs, then loops through them, showing the URLs of the blobs found. You can copy the URI from the command window and paste it into a browser to view the file.

```
//Listing contents of container
for (ListBlobItem blobItem : container.listBlobs()) {
 System.out.println("URI of blob is: " + blobItem.getUri());
}
```

### Download blobs

Download blobs to your local disk using [CloudBlob.DownloadToFile](#).

The following code downloads the blob uploaded in a previous section, adding a suffix of `_DOWNLOADED` to the blob name so you can see both files on local disk.

```
// Download blob. In most cases, you would have to retrieve the reference
// to cloudBlockBlob here. However, we created that reference earlier, and
// haven't changed the blob we're interested in, so we can reuse it.
// Here we are creating a new file to download to. Alternatively you can also pass in the path as a string
// into downloadToFile method: blob.downloadToFile("/path/to/new/file").
downloadedFile = new File(sourceFile.getParentFile(), "downloadedFile.txt");
blob.downloadToFile(downloadedFile.getAbsolutePath());
```

### Clean up resources

If you no longer need the blobs that you have uploaded, you can delete the entire container using [CloudBlobContainer.DeleteIfExists](#). This method also deletes the files in the container.

```
try {
if(container != null)
 container.deleteIfExists();
} catch (StorageException ex) {
System.out.println(String.format("Service error. Http code: %d and error code: %s", ex.getHttpStatusCode(),
ex.getErrorCode()));
}

System.out.println("Deleting the source, and downloaded files");

if(downloadedFile != null)
downloadedFile.deleteOnExit();

if(sourceFile != null)
sourceFile.deleteOnExit();
```

## Next steps

In this article, you learned how to transfer files between a local disk and Azure Blob storage using Java. To learn more about working with Java, continue to our GitHub source code repository.

[Java API Reference Code Samples for Java](#)

# Quickstart: Manage blobs with Python v12 SDK

8/22/2022 • 7 minutes to read • [Edit Online](#)

In this quickstart, you learn to manage blobs by using Python. Blobs are objects that can hold large amounts of text or binary data, including images, documents, streaming media, and archive data. You'll upload, download, and list blobs, and you'll create and delete containers.

More resources:

- [API reference documentation](#)
- [Library source code](#)
- [Package \(Python Package Index\)](#)
- [Samples](#)

## Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- An Azure Storage account. [Create a storage account](#).
- [Python](#) 2.7 or 3.6+.

## Setting up

This section walks you through preparing a project to work with the Azure Blob Storage client library v12 for Python.

### Create the project

Create a Python application named *blob-quickstart-v12*.

1. In a console window (such as cmd, PowerShell, or Bash), create a new directory for the project.

```
mkdir blob-quickstart-v12
```

2. Switch to the newly created *blob-quickstart-v12* directory.

```
cd blob-quickstart-v12
```

### Install the package

While still in the application directory, install the Azure Blob Storage client library for Python package by using the `pip install` command.

```
pip install azure-storage-blob
```

This command installs the Azure Blob Storage client library for Python package and all the libraries on which it depends. In this case, that is just the Azure core library for Python.

### Set up the app framework

From the project directory:

1. Open a new text file in your code editor

2. Add `import` statements

3. Create the structure for the program, including basic exception handling

Here's the code:

```
import os, uuid
from azure.storage.blob import BlobServiceClient, BlobClient, ContainerClient, __version__

try:
 print("Azure Blob Storage v" + __version__ + " - Python quickstart sample")

 # Quick start code goes here

except Exception as ex:
 print('Exception:')
 print(ex)
```

4. Save the new file as `blob-quickstart-v12.py` in the `blob-quickstart-v12` directory.

### Copy your credentials from the Azure portal

When the sample application makes a request to Azure Storage, it must be authorized. To authorize a request, add your storage account credentials to the application as a connection string. To view your storage account credentials, follow these steps:

1. Sign in to the [Azure portal](#).
2. Locate your storage account.
3. In the storage account menu pane, under **Security + networking**, select **Access keys**. Here, you can view the account access keys and the complete connection string for each key.

The screenshot shows the 'Access keys' page for a storage account named 'storagesamples'. On the left, there's a sidebar with options like 'Search (Ctrl+)', 'Show keys', 'Set rotation reminder', and 'Refresh'. The main area displays the 'Access keys' section, which includes a note about keeping keys secure in a vault. It shows one key pair: 'key1' with the value '<account-key>'. A red box highlights the 'Access keys' link in the sidebar.

4. In the **Access keys** pane, select **Show keys**.

5. In the **key1** section, locate the **Connection string** value. Select the **Copy to clipboard** icon to copy the connection string. You'll add the connection string value to an environment variable in the next section.

This screenshot shows the same 'Access keys' page as above, but with a red box highlighting the 'Connection string' field for 'key1'. This field contains the value '<connection-string>'.

### Configure your storage connection string

After you copy the connection string, write it to a new environment variable on the local machine running the application. To set the environment variable, open a console window, and follow the instructions for your operating system. Replace `<yourconnectionstring>` with your actual connection string.

- [Windows](#)
- [Linux and macOS](#)

```
setx AZURE_STORAGE_CONNECTION_STRING "<yourconnectionstring>"
```

After you add the environment variable in Windows, you must start a new instance of the command window.

#### **Restart programs**

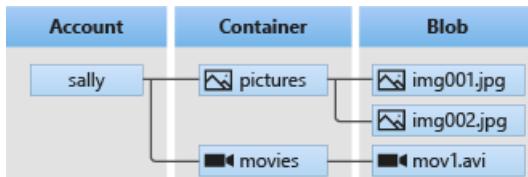
After you add the environment variable, restart any running programs that will need to read the environment variable. For example, restart your development environment or editor before you continue.

## Object model

Azure 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 offers three types of resources:

- The storage account
- A container in the storage account
- A blob in the container

The following diagram shows the relationship between these resources.



Use the following Python classes to interact with these resources:

- [BlobServiceClient](#): The `BlobServiceClient` class allows you to manipulate Azure Storage resources and blob containers.
- [ContainerClient](#): The `ContainerClient` class allows you to manipulate Azure Storage containers and their blobs.
- [BlobClient](#): The `BlobClient` class allows you to manipulate Azure Storage blobs.

## Code examples

These example code snippets show you how to do the following tasks with the Azure Blob Storage client library for Python:

- [Get the connection string](#)
- [Create a container](#)
- [Upload blobs to a container](#)
- [List the blobs in a container](#)
- [Download blobs](#)
- [Delete a container](#)

#### **Get the connection string**

The code below retrieves the storage account connection string from the environment variable created in the [Configure your storage connection string](#) section.

Add this code inside the `try` block:

```
Retrieve the connection string for use with the application. The storage
connection string is stored in an environment variable on the machine
running the application called AZURE_STORAGE_CONNECTION_STRING. If the environment variable is
created after the application is launched in a console or with Visual Studio,
the shell or application needs to be closed and reloaded to take the
environment variable into account.
connect_str = os.getenv('AZURE_STORAGE_CONNECTION_STRING')
```

## Create a container

Decide on a name for the new container. The code below appends a UUID value to the container name to ensure that it's unique.

### IMPORTANT

Container names must be lowercase. For more information about naming containers and blobs, see [Naming and Referencing Containers, Blobs, and Metadata](#).

Create an instance of the [BlobServiceClient](#) class by calling the [from\\_connection\\_string](#) method. Then, call the [create\\_container](#) method to actually create the container in your storage account.

Add this code to the end of the `try` block:

```
Create the BlobServiceClient object which will be used to create a container client
blob_service_client = BlobServiceClient.from_connection_string(connect_str)

Create a unique name for the container
container_name = str(uuid.uuid4())

Create the container
container_client = blob_service_client.create_container(container_name)
```

## Upload blobs to a container

The following code snippet:

1. Creates a local directory to hold data files.
2. Creates a text file in the local directory.
3. Gets a reference to a [BlobClient](#) object by calling the [get\\_blob\\_client](#) method on the [BlobServiceClient](#) from the [Create a container](#) section.
4. Uploads the local text file to the blob by calling the [upload\\_blob](#) method.

Add this code to the end of the `try` block:

```

Create a local directory to hold blob data
local_path = "./data"
os.mkdir(local_path)

Create a file in the local data directory to upload and download
local_file_name = str(uuid.uuid4()) + ".txt"
upload_file_path = os.path.join(local_path, local_file_name)

Write text to the file
file = open(upload_file_path, 'w')
file.write("Hello, World!")
file.close()

Create a blob client using the local file name as the name for the blob
blob_client = blob_service_client.get_blob_client(container=container_name, blob=local_file_name)

print("\nUploading to Azure Storage as blob:\n\t" + local_file_name)

Upload the created file
with open(upload_file_path, "rb") as data:
 blob_client.upload_blob(data)

```

## List the blobs in a container

List the blobs in the container by calling the [list\\_blobs](#) method. In this case, only one blob has been added to the container, so the listing operation returns just that one blob.

Add this code to the end of the `try` block:

```

print("\nListing blobs...")

List the blobs in the container
blob_list = container_client.list_blobs()
for blob in blob_list:
 print("\t" + blob.name)

```

## Download blobs

Download the previously created blob by calling the [download\\_blob](#) method. The example code adds a suffix of "DOWNLOAD" to the file name so that you can see both files in local file system.

Add this code to the end of the `try` block:

```

Download the blob to a local file
Add 'DOWNLOAD' before the .txt extension so you can see both files in the data directory
download_file_path = os.path.join(local_path, str.replace(local_file_name ,'.txt', 'DOWNLOAD.txt'))
blob_client = blob_service_client.get_container_client(container= container_name)
print("\nDownloading blob to \n\t" + download_file_path)

with open(download_file_path, "wb") as download_file:
 download_file.write(blob_client.download_blob(blob.name).readall())

```

## Delete a container

The following code cleans up the resources the app created by removing the entire container using the [delete\\_container](#) method. You can also delete the local files, if you like.

The app pauses for user input by calling `input()` before it deletes the blob, container, and local files. Verify that the resources were created correctly, before they're deleted.

Add this code to the end of the `try` block:

```
Clean up
print("\nPress the Enter key to begin clean up")
input()

print("Deleting blob container...")
container_client.delete_container()

print("Deleting the local source and downloaded files...")
os.remove(upload_file_path)
os.remove(download_file_path)
os.rmdir(local_path)

print("Done")
```

## Run the code

This app creates a test file in your local folder and uploads it to Azure Blob Storage. The example then lists the blobs in the container, and downloads the file with a new name. You can compare the old and new files.

Navigate to the directory containing the *blob-quickstart-v12.py* file, then execute the following `python` command to run the app.

```
python blob-quickstart-v12.py
```

The output of the app is similar to the following example:

```
Azure Blob Storage v12 - Python quickstart sample

Uploading to Azure Storage as blob:
 quickstartcf275796-2188-4057-b6fb-038352e35038.txt

Listing blobs...
 quickstartcf275796-2188-4057-b6fb-038352e35038.txt

Downloading blob to
 ./data/quickstartcf275796-2188-4057-b6fb-038352e35038DOWNLOAD.txt

Press the Enter key to begin clean up

Deleting blob container...
Deleting the local source and downloaded files...
Done
```

Before you begin the cleanup process, check your *data* folder for the two files. You can open them and observe that they're identical.

After you've verified the files, press the **Enter** key to delete the test files and finish the demo.

## Next steps

In this quickstart, you learned how to upload, download, and list blobs using Python.

To see Blob storage sample apps, continue to:

### [Azure Blob Storage SDK v12 Python samples](#)

- To learn more, see the [Azure Storage client libraries for Python](#).
- For tutorials, samples, quickstarts, and other documentation, visit [Azure for Python Developers](#).

# Quickstart: Manage blobs with Python v2.1 SDK

8/22/2022 • 6 minutes to read • [Edit Online](#)

In this quickstart, you learn to manage blobs by using Python. Blobs are objects that can hold large amounts of text or binary data, including images, documents, streaming media, and archive data. You'll upload, download, and list blobs, and you'll create and delete containers.

## NOTE

This quickstart uses a legacy version of the Azure Blob storage client library. To get started with the latest version, see [Quickstart: Manage blobs with Python v12 SDK](#).

## Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- An Azure Storage account. [Create a storage account](#).
- [Python](#).
- [Azure Storage SDK for Python](#).

## Download the sample application

The [sample application](#) in this quickstart is a basic Python application.

Use the following `git` command to download the application to your development environment.

```
git clone https://github.com/Azure-Samples/storage-blobs-python-quickstart.git
```

To review the Python program, open the `example.py` file at the root of the repository.

## Copy your credentials from the Azure portal

The sample application needs to authorize access to your storage account. Provide your storage account credentials to the application in the form of a connection string. To view your storage account credentials:

1. In to the [Azure portal](#) go to your storage account.
2. In the **Settings** section of the storage account overview, select **Access keys** to display your account access keys and connection string.
3. Note the name of your storage account, which you'll need for authorization.
4. Find the **Key** value under **key1**, and select **Copy** to copy the account key.



# Configure your storage connection string

In the application, provide your storage account name and account key to create a `BlockBlobService` object.

1. Open the `example.py` file from the Solution Explorer in your IDE.
2. Replace the `accountname` and `accountkey` values with your storage account name and key:

```
block_blob_service = BlockBlobService(
 account_name='accountname', account_key='accountkey')
```

3. Save and close the file.

## Run the sample

The sample program creates a test file in your *Documents* folder, uploads the file to Blob storage, lists the blobs in the file, and downloads the file with a new name.

1. Install the dependencies:

```
pip install azure-storage-blob==2.1.0
```

2. Go to the sample application:

```
cd storage-blobs-python-quickstart
```

3. Run the sample:

```
python example.py
```

You'll see messages similar to the following output:

```
Temp file = C:\Users\azureuser\Documents\QuickStart_9f4ed0f9-22d3-43e1-98d0-8b2c05c01078.txt

Uploading to Blob storage as blobQuickStart_9f4ed0f9-22d3-43e1-98d0-8b2c05c01078.txt

List blobs in the container
 Blob name: QuickStart_9f4ed0f9-22d3-43e1-98d0-8b2c05c01078.txt

Downloading blob to C:\Users\azureuser\Documents\QuickStart_9f4ed0f9-22d3-43e1-98d0-
8b2c05c01078_DOWNLOADED.txt
```

4. Before you continue, go to your *Documents* folder and check for the two files.

- `QuickStart_<universally-unique-identifier>`
- `QuickStart_<universally-unique-identifier>_DOWNLOADED`

5. You can open them and see they're the same.

You can also use a tool like the [Azure Storage Explorer](#). It's good for viewing the files in Blob storage. Azure Storage Explorer is a free cross-platform tool that lets you access your storage account info.

6. After you've looked at the files, press any key to finish the sample and delete the test files.

## Learn about the sample code

Now that you know what the sample does, open the `example.py` file to look at the code.

## Get references to the storage objects

In this section, you instantiate the objects, create a new container, and then set permissions on the container so the blobs are public. You'll call the container `quickstartblobs`.

```
Create the BlockBlobService that the system uses to call the Blob service for the storage account.
block_blob_service = BlockBlobService(
 account_name='accountname', account_key='accountkey')

Create a container called 'quickstartblobs'.
container_name = 'quickstartblobs'
block_blob_service.create_container(container_name)

Set the permission so the blobs are public.
block_blob_service.set_container_acl(
 container_name, public_access=PublicAccess.Container)
```

First, you create the references to the objects used to access and manage Blob storage. These objects build on each other, and each is used by the next one in the list.

- Instantiate the `BlockBlobService` object, which points to the Blob service in your storage account.
- Instantiate the `CloudBlobContainer` object, which represents the container you're accessing. The system uses containers to organize your blobs like you use folders on your computer to organize your files.

Once you have the Cloud Blob container, instantiate the `CloudBlockBlob` object that points to the specific blob that you're interested in. You can then upload, download, and copy the blob as you need.

### IMPORTANT

Container names must be lowercase. For more information about container and blob names, see [Naming and Referencing Containers, Blobs, and Metadata](#).

## Upload blobs to the container

Blob storage supports block blobs, append blobs, and page blobs. Block blobs can be as large as 4.7 TB, and can be anything from Excel spreadsheets to large video files. You can use append blobs for logging when you want to write to a file and then keep adding more information. Page blobs are primarily used for the Virtual Hard Disk (VHD) files that back infrastructure as a service virtual machines (IaaS VMs). Block blobs are the most commonly used. This quickstart uses block blobs.

To upload a file to a blob, get the full file path by joining the directory name with the file name on your local drive. You can then upload the file to the specified path using the `create_blob_from_path` method.

The sample code creates a local file the system uses for the upload and download, storing the file the system uploads as `full_path_to_file` and the name of the blob as `local_file_name`. This example uploads the file to your container called `quickstartblobs`:

```

Create a file in Documents to test the upload and download.
local_path = os.path.expanduser("~/Documents")
local_file_name = "QuickStart_" + str(uuid.uuid4()) + ".txt"
full_path_to_file = os.path.join(local_path, local_file_name)

Write text to the file.
file = open(full_path_to_file, 'w')
file.write("Hello, World!")
file.close()

print("Temp file = " + full_path_to_file)
print("\nUploading to Blob storage as blob" + local_file_name)

Upload the created file, use local_file_name for the blob name.
block_blob_service.create_blob_from_path(
 container_name, local_file_name, full_path_to_file)

```

There are several upload methods that you can use with Blob storage. For example, if you have a memory stream, you can use the `create_blob_from_stream` method rather than `create_blob_from_path`.

### List the blobs in a container

The following code creates a `generator` for the `list_blobs` method. The code loops through the list of blobs in the container and prints their names to the console.

```

List the blobs in the container.
print("\nList blobs in the container")
generator = block_blob_service.list_blobs(container_name)
for blob in generator:
 print("\t Blob name: " + blob.name)

```

### Download the blobs

Download blobs to your local disk using the `get_blob_to_path` method. The following code downloads the blob you uploaded previously. The system appends `_DOWNLOADED` to the blob name so you can see both files on your local disk.

```

Download the blob(s).
Add '_DOWNLOADED' as prefix to '.txt' so you can see both files in Documents.
full_path_to_file2 = os.path.join(local_path, local_file_name.replace(
 '.txt', '_DOWNLOADED.txt'))
print("\nDownloading blob to " + full_path_to_file2)
block_blob_service.get_blob_to_path(
 container_name, local_file_name, full_path_to_file2)

```

### Clean up resources

If you no longer need the blobs uploaded in this quickstart, you can delete the entire container using the `delete_container` method. To delete individual files instead, use the `delete_blob` method.

```

Clean up resources. This includes the container and the temp files.
block_blob_service.delete_container(container_name)
os.remove(full_path_to_file)
os.remove(full_path_to_file2)

```

## Resources for developing Python applications with blobs

For more about Python development with Blob storage, see these additional resources:

## **Binaries and source code**

- View, download, and install the [Python client library source code](#) for Azure Storage on GitHub.

## **Client library reference and samples**

- For more about the Python client library, see the [Azure Storage libraries for Python](#).
- Explore [Blob storage samples](#) written using the Python client library.

## **Next steps**

In this quickstart, you learned how to transfer files between a local disk and Azure Blob storage using Python.

For more about the Storage Explorer and Blobs, see [Manage Azure Blob storage resources with Storage Explorer](#).

# Quickstart: Manage blobs with JavaScript v12 SDK in Node.js

8/22/2022 • 7 minutes to read • [Edit Online](#)

In this quickstart, you learn to manage blobs by using Node.js. Blobs are objects that can hold large amounts of text or binary data, including images, documents, streaming media, and archive data.

These [example code](#) snippets show you how to perform the following with the Azure Blob storage package library for JavaScript:

- [Get the connection string](#)
- [Create a container](#)
- [Upload blobs to a container](#)
- [List the blobs in a container](#)
- [Download blobs](#)
- [Delete a container](#)

Additional resources:

[API reference](#) | [Library source code](#) | [Package \(npm\)](#) | [Samples](#)

## Prerequisites

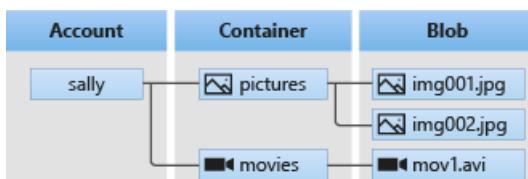
- An Azure account with an active subscription. [Create an account for free](#).
- An Azure Storage account. [Create a storage account](#).
- [Node.js LTS](#).
- [Microsoft Visual Studio Code](#)

## Object model

Azure 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 offers three types of resources:

- The storage account
- A container in the storage account
- A blob in the container

The following diagram shows the relationship between these resources.



Use the following JavaScript classes to interact with these resources:

- [BlobServiceClient](#): The `BlobServiceClient` class allows you to manipulate Azure Storage resources and blob containers.
- [ContainerClient](#): The `ContainerClient` class allows you to manipulate Azure Storage containers and their

blobs.

- **BlobClient**: The `BlobClient` class allows you to manipulate Azure Storage blobs.

## Create the Node.js project

Create a JavaScript application named *blob-quickstart-v12*.

1. In a console window (such as cmd, PowerShell, or Bash), create a new directory for the project.

```
mkdir blob-quickstart-v12
```

2. Switch to the newly created *blob-quickstart-v12* directory.

```
cd blob-quickstart-v12
```

3. Create a *package.json*.

```
npm init -y
```

4. Open the project in Visual Studio Code:

```
code .
```

## Install the npm package for blob storage

1. Install the Azure Storage npm package:

```
npm install @azure/storage-blob
```

2. Install other dependencies used in this quickstart:

```
npm install uuid dotenv
```

## Create JavaScript file

From the project directory:

1. Create a new file named `index.js`.
2. Copy the following code into the file. More code will be added as you go through this quickstart.

```

const { BlobServiceClient } = require('@azure/storage-blob');
const { v1: uuidv1 } = require('uuid');
require('dotenv').config()

async function main() {
 console.log('Azure Blob storage v12 - JavaScript quickstart sample');

 // Quick start code goes here

}

main()
.then(() => console.log('Done'))
.catch((ex) => console.log(ex.message));

```

## Copy your credentials from the Azure portal

When the sample application makes a request to Azure Storage, it must be authorized. To authorize a request, add your storage account credentials to the application as a connection string. To view your storage account credentials, follow these steps:

1. Sign in to the [Azure portal](#).
2. Locate your storage account.
3. In the storage account menu pane, under **Security + networking**, select **Access keys**. Here, you can view the account access keys and the complete connection string for each key.

The screenshot shows the 'Access keys' page for a storage account named 'storagesamples'. The 'Access keys' section is highlighted with a red box. The 'key1' row shows the 'Connection string' field with a copy icon highlighted by a red box.

4. In the **Access keys** pane, select **Show keys**.
5. In the **key1** section, locate the **Connection string** value. Select the **Copy to clipboard** icon to copy the connection string. You'll add the connection string value to an environment variable in the next section.

The screenshot shows the 'Access keys' page for a storage account named 'storagesamples'. The 'key1' section is highlighted with a red box. The 'Connection string' field is highlighted with a red box, and its copy icon is also highlighted with a red box.

## Configure your storage connection string

After you copy the connection string, write it to a new environment variable on the local machine running the application. To set the environment variable, open a console window, and follow the instructions for your operating system. Replace <yourconnectionstring> with your actual connection string.

- [Windows](#)
- [Linux and macOS](#)

```
setx AZURE_STORAGE_CONNECTION_STRING "<yourconnectionstring>"
```

After you add the environment variable in Windows, you must start a new instance of the command window.

#### Restart programs

After you add the environment variable, restart any running programs that will need to read the environment variable. For example, restart your development environment or editor before you continue.

## Get the connection string

The code below retrieves the connection string for the storage account from the environment variable created in the [Configure your storage connection string](#) section.

Add this code inside the `main` function:

```
const AZURE_STORAGE_CONNECTION_STRING =
 process.env.AZURE_STORAGE_CONNECTION_STRING;

if (!AZURE_STORAGE_CONNECTION_STRING) {
 throw Error("Azure Storage Connection string not found");
}
```

## Create a container

1. Decide on a name for the new container. Container names must be lowercase.

For more information about naming containers and blobs, see [Naming and Referencing Containers, Blobs, and Metadata](#).

2. Add this code to the end of the `main` function:

```
// Create the BlobServiceClient object which will be used to create a container client
const blobServiceClient = BlobServiceClient.fromConnectionString(
 AZURE_STORAGE_CONNECTION_STRING
);

// Create a unique name for the container
const containerName = "quickstart" + uuidv1();

console.log("\nCreating container...");
console.log("\t", containerName);

// Get a reference to a container
const containerClient = blobServiceClient.getContainerClient(containerName);
// Create the container
const createContainerResponse = await containerClient.create();
console.log(
 "Container was created successfully. requestId: ",
 createContainerResponse.requestId
)
```

The preceding code creates an instance of the `BlobServiceClient` class by calling the `fromConnectionString` method. Then, call the `getContainerClient` method to get a reference to a container. Finally, call `create` to actually create the container in your storage account.

## Upload blobs to a container

Copy the following code to the end of the `main` function to upload a text string to a blob:

```

// Create a unique name for the blob
const blobName = "quickstart" + uuidv1() + ".txt";

// Get a block blob client
const blockBlobClient = containerClient.getBlockBlobClient(blobName);

console.log("\nUploading to Azure storage as blob:\n\t", blobName);

// Upload data to the blob
const data = "Hello, World!";
const uploadBlobResponse = await blockBlobClient.upload(data, data.length);
console.log(
 "Blob was uploaded successfully. requestId: ",
 uploadBlobResponse.requestId
);

```

The preceding code gets a reference to a [BlockBlobClient](#) object by calling the [getBlockBlobClient](#) method on the [ContainerClient](#) from the [Create a container](#) section. The code uploads the text string data to the blob by calling the [upload](#) method.

## List the blobs in a container

Add the following code to the end of the `main` function to list the blobs in the container.

```

console.log("\nListing blobs...");

// List the blob(s) in the container.
for await (const blob of containerClient.listBlobsFlat()) {
 console.log("\t", blob.name);
}

```

The preceding code calls the [listBlobsFlat](#) method. In this case, only one blob has been added to the container, so the listing operation returns just that one blob.

## Download blobs

1. Add the following code to the end of the `main` function to download the previously created blob into the app runtime.

```

// Get blob content from position 0 to the end
// In Node.js, get downloaded data by accessing downloadBlockBlobResponse.readableStreamBody
// In browsers, get downloaded data by accessing downloadBlockBlobResponse.blobBody
const downloadBlockBlobResponse = await blockBlobClient.download(0);
console.log("\nDownloaded blob content...");
console.log(
 "\t",
 await streamToString(downloadBlockBlobResponse.readableStreamBody)
);

```

The preceding code calls the [download](#) method.

2. Copy the following code *after* the `main` function to convert a stream back into a string.

```
// Convert stream to text
async function streamToText(readable) {
 readable.setEncoding('utf8');
 let data = '';
 for await (const chunk of readable) {
 data += chunk;
 }
 return data;
}
```

## Delete a container

Add this code to the end of the `main` function to delete the container and all its blobs:

```
// Delete container
console.log("\nDeleting container...");

const deleteContainerResponse = await containerClient.delete();
console.log(
 "Container was deleted successfully. requestId: ",
 deleteContainerResponse.requestId
);
```

The preceding code cleans up the resources the app created by removing the entire container using the [delete](#) method. You can also delete the local files, if you like.

## Run the code

1. From a Visual Studio Code terminal, run the app.

```
node index.js
```

2. The output of the app is similar to the following example:

```
Azure Blob storage v12 - JavaScript quickstart sample

Creating container...
quickstart4a0780c0-fb72-11e9-b7b9-b387d3c488da

Uploading to Azure Storage as blob:
quickstart4a3128d0-fb72-11e9-b7b9-b387d3c488da.txt

Listing blobs...
quickstart4a3128d0-fb72-11e9-b7b9-b387d3c488da.txt

Downloaded blob content...
Hello, World!

Deleting container...
Done
```

Step through the code in your debugger and check your [Azure portal](#) throughout the process. Check to see that the container is being created. You can open the blob inside the container and view the contents.

## Use the storage emulator

This quickstart created a container and blob on the Azure cloud. You can also use the Azure Blob storage npm

package to create these resources locally on the [Azure Storage emulator](#) for development and testing.

## Clean up

1. When you're done with this quickstart, delete the `blob-quickstart-v12` directory.
2. If you're done using your Azure Storage resource, use the [Azure CLI to remove the Storage resource](#).

## Next steps

In this quickstart, you learned how to upload, download, and list blobs using JavaScript.

For tutorials, samples, quickstarts, and other documentation, visit:

[Azure for JavaScript developer center](#)

- To learn how to deploy a web app that uses Azure Blob storage, see [Tutorial: Upload image data in the cloud with Azure Storage](#)
- To see Blob storage sample apps, continue to [Azure Blob storage package library v12 JavaScript samples](#).
- To learn more, see the [Azure Blob storage client library for JavaScript](#).

# Quickstart: Manage blobs with JavaScript v12 SDK in a browser

8/22/2022 • 10 minutes to read • [Edit Online](#)

Azure Blob storage is optimized for storing large amounts of unstructured data. Blobs are objects that can hold text or binary data, including images, documents, streaming media, and archive data. In this quickstart, you learn to manage blobs by using JavaScript in a browser. You'll upload and list blobs, and you'll create and delete containers.

The [example code](#) shows you how to accomplish the following tasks with the Azure Blob storage client library for JavaScript:

- [Declare fields for UI elements](#)
- [Add your storage account info](#)
- [Create client objects](#)
- [Create and delete a storage container](#)
- [List blobs](#)
- [Upload blobs](#)
- [Delete blobs](#)

Additional resources:

[API reference](#) | [Library source code](#) | [Package \(npm\)](#) | [Samples](#)

## Prerequisites

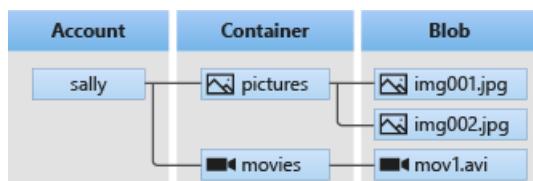
- [An Azure account with an active subscription](#)
- [An Azure Storage account](#)
- [Node.js LTS](#)
- [Microsoft Visual Studio Code](#)

## Object model

Blob storage offers three types of resources:

- The storage account
- A container in the storage account
- A blob in the container

The following diagram shows the relationship between these resources.



In this quickstart, you'll use the following JavaScript classes to interact with these resources:

- [BlobServiceClient](#): The `BlobServiceClient` class allows you to manipulate Azure Storage resources and blob containers.

- **ContainerClient**: The `ContainerClient` class allows you to manipulate Azure Storage containers and their blobs.
- **BlockBlobClient**: The `BlockBlobClient` class allows you to manipulate Azure Storage blobs.

## Configure storage account for browser access

To programmatically access your storage account from a web browser, you need to configure CORS access and create an SAS connection string.

### Create a CORS rule

Before your web application can access blob storage from the client, you must configure your account to enable [cross-origin resource sharing](#), or CORS.

In the Azure portal, select your storage account. To define a new CORS rule, navigate to the **Settings** section and select CORS. For this quickstart, you create a fully-open CORS rule:

ALLOWED ORIGINS	ALLOWED METHODS	ALLOWED HEADERS	EXPOSED HEADERS	MAX AGE
*	DELETE, GET, HEAD, MERGE, POST, OPTIONS, and PUT	*	*	86400

The following table describes each CORS setting and explains the values used to define the rule.

SETTING	VALUE	DESCRIPTION
ALLOWED ORIGINS	*	Accepts a comma-delimited list of domains set as acceptable origins. Setting the value to <code>*</code> allows all domains access to the storage account.
ALLOWED METHODS	DELETE, GET, HEAD, MERGE, POST, OPTIONS, and PUT	Lists the HTTP verbs allowed to execute against the storage account. For the purposes of this quickstart, select all available options.

SETTING	VALUE	DESCRIPTION
ALLOWED HEADERS	*	Defines a list of request headers (including prefixed headers) allowed by the storage account. Setting the value to <code>*</code> allows all headers access.
EXPOSED HEADERS	*	Lists the allowed response headers by the account. Setting the value to <code>*</code> allows the account to send any header.
MAX AGE	86400	The maximum amount of time the browser caches the preflight OPTIONS request in seconds. A value of <code>86400</code> allows the cache to remain for a full day.

After you fill in the fields with the values from this table, select the **Save** button.

#### IMPORTANT

Ensure any settings you use in production expose the minimum amount of access necessary to your storage account to maintain secure access. The CORS settings described here are appropriate for a quickstart as it defines a lenient security policy. These settings, however, are not recommended for a real-world context.

### Create a SAS connection string

The shared access signature (SAS) is used by code running in the browser to authorize Azure Blob storage requests. By using the SAS, the client can authorize access to storage resources without the account access key or connection string. For more information on SAS, see [Using shared access signatures \(SAS\)](#).

Follow these steps to get the Blob service SAS URL:

1. In the Azure portal, select your storage account.
2. Navigate to the **Security + networking** section and select **Shared access signature**.
3. Review the **Allowed services** to understand the SAS token will have access to all of your storage account services:
  - Blob
  - File
  - Queue
  - Table
4. Select the **Allowed resources types** to include:
  - Service
  - Container
  - Object
5. Review the **Start and expiry date/time** to understand the SAS token has a limited lifetime by default.
6. Scroll down and select the **Generate SAS and connection string** button.
7. Scroll down further and locate the **Blob service SAS URL** field
8. Select the **Copy to clipboard** button at the far-right end of the **Blob service SAS URL** field.
9. Save the copied URL somewhere for use in an upcoming step.

### Create the JavaScript project

Create a JavaScript application named `blob-quickstart-v12`.

1. In a console window (such as cmd, PowerShell, or Bash), create a new directory for the project.

```
mkdir blob-quickstart-v12
```

2. Switch to the newly created *blob-quickstart-v12* directory.

```
cd blob-quickstart-v12
```

3. Create a *package.json*.

```
npm init -y
```

4. Open the project in Visual Studio Code:

```
code .
```

## Install the npm package for blob storage

1. In a Visual Studio Code terminal, install the Azure Storage npm package:

```
npm install @azure/storage-blob
```

2. Install a bundler package to bundle the files and package for the browser:

```
npm install parcel
```

If you plan to use a different bundler, learn more about [bundling the Azure SDK](#).

## Configure browser bundling

1. In Visual Studio Code, open the *package.json* file and add a `browserlist`. This `browserlist` targets the latest version of popular browsers. The full *package.json* file should now look like this:

```
"browserslist": [
 "last 1 Edge version",
 "last 1 Chrome version",
 "last 1 Firefox version",
 "last 1 safari version",
 "last 1 webkit version"
,
```

2. Add a `start` script to bundle the website:

```
"scripts": {
 "start": "parcel ./index.html"
,
```

## Create the HTML file

1. Create `index.html` and add the following HTML code:

```

<!-- index.html -->
<!DOCTYPE html>
<html>

<body>
 <button id="create-container-button">Create container</button>
 <button id="select-button">Select and upload files</button>
 <input type="file" id="file-input" multiple style="display: none;" />
 <button id="list-button">List files</button>
 <button id="delete-button">Delete selected files</button>
 <button id="delete-container-button">Delete container</button>
 <p>Status:</p>
 <p id="status" style="height:160px; width: 593px; overflow: scroll;" />
 <p>Files:</p>
 <select id="file-list" multiple style="height:222px; width: 593px; overflow: scroll;" />
</body>

<script type="module" src=".//index.js"></script>

</html>

```

## Create the JavaScript file

From the project directory:

1. Create a new file named `index.js`.
2. Add the Azure Storage npm package.

```
const { BlobServiceClient } = require("@azure/storage-blob");
```

## Declare fields for UI elements

Add DOM elements for user interaction:

```

const createContainerButton = document.getElementById("create-container-button");
const deleteContainerButton = document.getElementById("delete-container-button");
const selectButton = document.getElementById("select-button");
const fileInput = document.getElementById("file-input");
const listButton = document.getElementById("list-button");
const deleteButton = document.getElementById("delete-button");
const status = document.getElementById("status");
const fileList = document.getElementById("file-list");

const reportStatus = message => {
 status.innerHTML += `${message}
`;
 status.scrollTop = status.scrollHeight;
}

```

This code declares fields for each HTML element and implements a `reportStatus` function to display output.

## Add your storage account info

Add the following code at the end of the `index.js` file to access your storage account. Replace the `<placeholder>` with your Blob service SAS URL that you generated earlier. Add the following code to the end of the `index.js` file.

```
// Update <placeholder> with your Blob service SAS URL string
const blobSasUrl = "<placeholder>";
```

## Create client objects

Create [BlobServiceClient](#) and [ContainerClient](#) objects to connect to your storage account. Add the following code to the end of the *index.js* file.

```
// Create a new BlobServiceClient
const blobServiceClient = new BlobServiceClient(blobSasUrl);

// Create a unique name for the container by
// appending the current time to the file name
const containerName = "container" + new Date().getTime();

// Get a container client from the BlobServiceClient
const containerClient = blobServiceClient.getContainerClient(containerName);
```

## Create and delete a storage container

Create and delete the storage container when you select the corresponding button on the web page. Add the following code to the end of the *index.js* file.

```
const createContainer = async () => {
 try {
 reportStatus(`Creating container "${containerName}"...`);
 await containerClient.create();
 reportStatus(`Done.`);
 } catch (error) {
 reportStatus(error.message);
 }
};

const deleteContainer = async () => {
 try {
 reportStatus(`Deleting container "${containerName}"...`);
 await containerClient.delete();
 reportStatus(`Done.`);
 } catch (error) {
 reportStatus(error.message);
 }
};

createContainerButton.addEventListener("click", createContainer);
deleteContainerButton.addEventListener("click", deleteContainer);
```

## List blobs

List the contents of the storage container when you select the **List files** button. Add the following code to the end of the *index.js* file.

```

const listFiles = async () => {
 fileList.size = 0;
 fileList.innerHTML = "";
 try {
 reportStatus("Retrieving file list...");
 let iter = containerClient.listBlobsFlat();
 let blobItem = await iter.next();
 while (!blobItem.done) {
 fileList.size += 1;
 fileList.innerHTML += `<option>${blobItem.value.name}</option>`;
 blobItem = await iter.next();
 }
 if (fileList.size > 0) {
 reportStatus("Done.");
 } else {
 reportStatus("The container does not contain any files.");
 }
 } catch (error) {
 reportStatus(error.message);
 }
};

listButton.addEventListener("click", listFiles);

```

This code calls the `ContainerClient.listBlobsFlat` function, then uses an iterator to retrieve the name of each `BlobItem` returned. For each `BlobItem`, it updates the `Files` list with the `name` property value.

## Upload blobs to a container

Upload files to the storage container when you select the **Select and upload files** button. Add the following code to the end of the `index.js` file.

```

const uploadFiles = async () => {
 try {
 reportStatus("Uploading files...");
 const promises = [];
 for (const file of fileInput.files) {
 const blockBlobClient = containerClient.getBlockBlobClient(file.name);
 promises.push(blockBlobClient.uploadBrowserData(file));
 }
 await Promise.all(promises);
 reportStatus("Done.");
 listFiles();
 }
 catch (error) {
 reportStatus(error.message);
 }
};

selectButton.addEventListener("click", () => fileInput.click());
fileInput.addEventListener("change", uploadFiles);

```

This code connects the **Select and upload files** button to the hidden `file-input` element. The button `click` event triggers the file input `click` event and displays the file picker. After you select files and close the dialog box, the `input` event occurs and the `uploadFiles` function is called. This function creates a `BlockBlobClient` object, then calls the browser-only `uploadBrowserData` function for each file you selected. Each call returns a `Promise`. Each `Promise` is added to a list so that they can all be awaited together, causing the files to upload in parallel.

## Delete blobs

Delete files from the storage container when you select the **Delete selected files** button. Add the following code to the end of the *index.js* file.

```
const deleteFiles = async () => {
 try {
 if (fileList.selectedOptions.length > 0) {
 reportStatus("Deleting files...");
 for (const option of fileList.selectedOptions) {
 await containerClient.deleteBlob(option.text);
 }
 reportStatus("Done.");
 listFiles();
 } else {
 reportStatus("No files selected.");
 }
 } catch (error) {
 reportStatus(error.message);
 }
};

deleteButton.addEventListener("click", deleteFiles);
```

This code calls the [ContainerClient.deleteBlob](#) function to remove each file selected in the list. It then calls the [listFiles](#) function shown earlier to refresh the contents of the **Files** list.

## Run the code

1. From a Visual Studio Code terminal, run the app.

```
npm start
```

This process bundles the files and starts a web server.

2. Access the web site with a browser using the following URL:

```
http://localhost:1234
```

## Step 1 - Create a container

1. In the web app, select **Create container**. The status indicates that a container was created.
2. In the Azure portal, verify your container was created. Select your storage account. Under **Blob service**, select **Containers**. Verify that the new container appears. (You may need to select **Refresh**.)

## Step 2 - Upload a blob to the container

1. On your local computer, create and save a test file, such as *test.txt*.
2. In the web app, select **Select and upload files**.
3. Browse to your test file, and then select **Open**. The status indicates that the file was uploaded, and the file list was retrieved.
4. In the Azure portal, select the name of the new container that you created earlier. Verify that the test file appears.

## Step 3 - Delete the blob

1. In the web app, under **Files**, select the test file.

2. Select **Delete selected files**. The status indicates that the file was deleted and that the container contains no files.
3. In the Azure portal, select **Refresh**. Verify that you see **No blobs found**.

## Step 4 - Delete the container

1. In the web app, select **Delete container**. The status indicates that the container was deleted.
2. In the Azure portal, select the <account-name> | **Containers** link at the top-left of the portal pane.
3. Select **Refresh**. The new container disappears.
4. Close the web app.

## Use the storage emulator

This quickstart created a container and blob on the Azure cloud. You can also use the Azure Blob storage npm package to create these resources locally on the [Azure Storage emulator](#) for development and testing.

## Clean up resources

1. When you're done with this quickstart, delete the `blob-quickstart-v12` directory.
2. If you're done using your Azure Storage resource, remove your resource group using either method:
  - Use the [Azure CLI to remove the Storage resource](#)
  - Use the [Azure portal to remove the resource](#).

## Next steps

In this quickstart, you learned how to upload, list, and delete blobs using JavaScript. You also learned how to create and delete a blob storage container.

For tutorials, samples, quickstarts, and other documentation, visit:

### [Azure for JavaScript documentation](#)

- To learn more, see the [Azure Blob storage client library for JavaScript](#).
- To see Blob storage sample apps, continue to [Azure Blob storage client library v12 JavaScript samples](#).

# Quickstart: Azure Blob Storage client library v12 for C++

8/22/2022 • 6 minutes to read • [Edit Online](#)

Get started with the Azure Blob Storage client library v12 for C++. Azure Blob Storage is Microsoft's object storage solution for the cloud. Follow steps to install the package and try out example code for basic tasks. Blob Storage is optimized for storing massive amounts of unstructured data.

Use the Azure Blob Storage client library v12 for C++ to:

- Create a container
- Upload a blob to Azure Storage
- List all of the blobs in a container
- Download the blob to your local computer
- Delete a container

Resources:

- [API reference documentation](#)
- [Library source code](#)
- [Samples](#)

## Prerequisites

- [Azure subscription](#)
- [Azure storage account](#)
- [C++ compiler](#)
- [CMake](#)
- [Vcpkg - C and C++ package manager](#)

## Setting up

This section walks you through preparing a project to work with the Azure Blob Storage client library v12 for C++.

### Install the packages

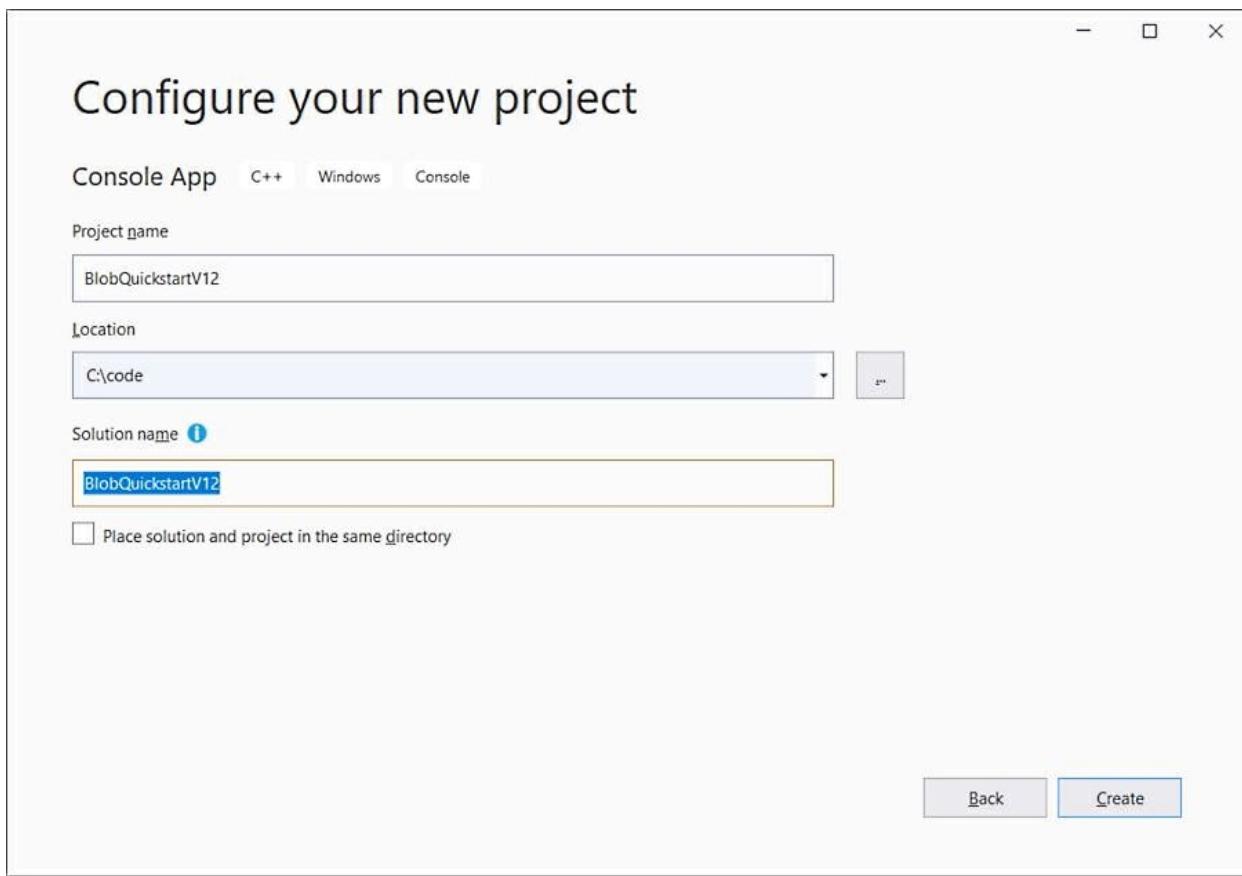
The `vcpkg install` command will install the Azure Storage Blobs SDK for C++ and necessary dependencies:

```
vcpkg.exe install azure-storage-blobs-cpp:x64-windows
```

For more information, visit GitHub to acquire and build the [Azure SDK for C++](#).

### Create the project

In Visual Studio, create a new C++ console application for Windows called *BlobQuickstartV12*.



## Copy your credentials from the Azure portal

When the sample application makes a request to Azure Storage, it must be authorized. To authorize a request, add your storage account credentials to the application as a connection string. To view your storage account credentials, follow these steps:

1. Sign in to the [Azure portal](#).
2. Locate your storage account.
3. In the storage account menu pane, under **Security + networking**, select **Access keys**. Here, you can view the account access keys and the complete connection string for each key.

The screenshot shows the 'storagesamples' storage account in the Azure portal. The 'Access keys' tab is selected and highlighted with a red box. The 'Show keys' button is visible above the key details. The storage account name is listed as 'storagesamples'.

4. In the **Access keys** pane, select **Show keys**.
5. In the **key1** section, locate the **Connection string** value. Select the **Copy to clipboard** icon to copy the connection string. You'll add the connection string value to an environment variable in the next section.

The screenshot shows the 'Access keys' pane for the 'key1' section. The 'Connection string' field is selected and highlighted with a red box, indicating it is ready to be copied.

## Configure your storage connection string

After you copy the connection string, write it to a new environment variable on the local machine running the application. To set the environment variable, open a console window, and follow the instructions for your operating system. Replace `<yourconnectionstring>` with your actual connection string.

- [Windows](#)
- [Linux and macOS](#)

```
setx AZURE_STORAGE_CONNECTION_STRING "<yourconnectionstring>"
```

After you add the environment variable in Windows, you must start a new instance of the command window.

### Restart programs

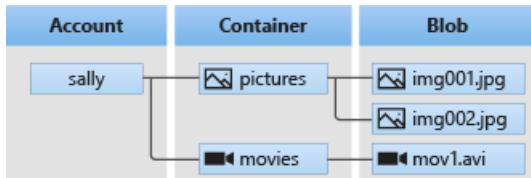
After you add the environment variable, restart any running programs that will need to read the environment variable. For example, restart your development environment or editor before you continue.

## Object model

Azure 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 offers three types of resources:

- The storage account
- A container in the storage account
- A blob in the container

The following diagram shows the relationship between these resources.



Use these C++ classes to interact with these resources:

- **BlobServiceClient**: The `BlobServiceClient` class allows you to manipulate Azure Storage resources and blob containers.
- **BlobContainerClient**: The `BlobContainerClient` class allows you to manipulate Azure Storage containers and their blobs.
- **BlobClient**: The `BlobClient` class allows you to manipulate Azure Storage blobs. It's the base class for all specialized blob classes.
- **BlockBlobClient**: The `BlockBlobClient` class allows you to manipulate Azure Storage block blobs.

## Code examples

These example code snippets show you how to do the following tasks with the Azure Blob Storage client library for C++:

- [Add include files](#)
- [Get the connection string](#)
- [Create a container](#)
- [Upload blobs to a container](#)
- [List the blobs in a container](#)

- [Download blobs](#)
- [Delete a container](#)

## Add include files

From the project directory:

1. Open the *BlobQuickstartV12.sln* solution file in Visual Studio
2. Inside Visual Studio, open the *BlobQuickstartV12.cpp* source file
3. Remove any code inside `main` that was autogenerated
4. Add `#include` statements

```
#include <stdlib.h>
#include <iostream>
#include <azure/storage/blobs.hpp>
```

## Get the connection string

The code below retrieves the connection string for your storage account from the environment variable created in [Configure your storage connection string](#).

Add this code inside `main()`:

```
// Retrieve the connection string for use with the application. The storage
// connection string is stored in an environment variable on the machine
// running the application called AZURE_STORAGE_CONNECTION_STRING.
// Note that _MSC_VER is set when using MSVC compiler.
static const char* AZURE_STORAGE_CONNECTION_STRING = "AZURE_STORAGE_CONNECTION_STRING";
#ifndef _MSC_VER
 const char* connectionString = std::getenv(AZURE_STORAGE_CONNECTION_STRING);
#else
 // Use getenv_s for MSVC
 size_t requiredSize;
 getenv_s(&requiredSize, NULL, NULL, AZURE_STORAGE_CONNECTION_STRING);
 if (requiredSize == 0) {
 throw std::runtime_error("missing connection string from env.");
 }
 std::vector<char> value(requiredSize);
 getenv_s(&requiredSize, value.data(), value.size(), AZURE_STORAGE_CONNECTION_STRING);
 std::string connectionStringStr = std::string(value.begin(), value.end());
 const char* connectionString = connectionStringStr.c_str();
#endif
```

## Create a container

Create an instance of the [BlobContainerClient](#) class by calling the [CreateFromConnectionString](#) function. Then call [CreateIfNotExists](#) to create the actual container in your storage account.

### IMPORTANT

Container names must be lowercase. For more information about naming containers and blobs, see [Naming and Referencing Containers, Blobs, and Metadata](#).

Add this code to the end of `main()`:

```

using namespace Azure::Storage::Blobs;

std::string containerName = "myblobcontainer";

// Initialize a new instance of BlobContainerClient
BlobContainerClient containerClient
 = BlobContainerClient::CreateFromConnectionString(connectionString, containerName);

// Create the container. This will do nothing if the container already exists.
std::cout << "Creating container: " << containerName << std::endl;
containerClient.CreateIfNotExists();

```

## Upload blobs to a container

The following code snippet:

1. Declares a string containing "Hello Azure!".
2. Gets a reference to a [BlockBlobClient](#) object by calling [GetBlockBlobClient](#) on the container from the [Create a container](#) section.
3. Uploads the string to the blob by calling the [UploadFrom](#) function. This function creates the blob if it doesn't already exist, or updates it if it does.

Add this code to the end of `main()`:

```

std::string blobName = "blob.txt";
uint8_t blobContent[] = "Hello Azure!";
// Create the block blob client
BlockBlobClient blobClient = containerClient.GetBlockBlobClient(blobName);

// Upload the blob
std::cout << "Uploading blob: " << blobName << std::endl;
blobClient.UploadFrom(blobContent, sizeof(blobContent));

```

## List the blobs in a container

List the blobs in the container by calling the [ListBlobs](#) function. Only one blob has been added to the container, so the operation returns just that blob.

Add this code to the end of `main()`:

```

std::cout << "Listing blobs..." << std::endl;
auto listBlobsResponse = containerClient.ListBlobs();
for (auto blobItem : listBlobsResponse.Blobs)
{
 std::cout << "Blob name: " << blobItem.Name << std::endl;
}

```

## Download blobs

Get the properties of the uploaded blob. Then, declare and resize a new `std::vector<uint8_t>` object by using the properties of the uploaded blob. Download the previously created blob into the new `std::vector<uint8_t>` object by calling the [DownloadTo](#) function in the [BlobClient](#) base class. Finally, display the downloaded blob data.

Add this code to the end of `main()`:

```
auto properties = blobClient.GetProperties().Value;
std::vector<uint8_t> downloadedBlob(properties.BlobSize);

blobClient.DownloadTo(downloadedBlob.data(), downloadedBlob.size());
std::cout << "Downloaded blob contents: " << std::string(downloadedBlob.begin(), downloadedBlob.end()) <<
std::endl;
```

## Delete a Blob

The following code deletes the blob from the Azure Blob Storage container by calling the [BlobClient.Delete](#) function.

```
std::cout << "Deleting blob: " << blobName << std::endl;
blobClient.Delete();
```

## Delete a container

The following code cleans up the resources the app created by deleting the entire container by using [BlobContainerClient.Delete](#).

Add this code to the end of `main()`:

```
std::cout << "Deleting container: " << containerName << std::endl;
containerClient.Delete();
```

## Run the code

This app creates a container and uploads a text file to Azure Blob Storage. The example then lists the blobs in the container, downloads the file, and displays the file contents. Finally, the app deletes the blob and the container.

The output of the app is similar to the following example:

```
Azure Blob Storage v12 - C++ quickstart sample
Creating container: myblobcontainer
Uploading blob: blob.txt
Listing blobs...
Blob name: blob.txt
Downloaded blob contents: Hello Azure!
Deleting blob: blob.txt
Deleting container: myblobcontainer
```

## Next steps

In this quickstart, you learned how to upload, download, and list blobs using C++. You also learned how to create and delete an Azure Blob Storage container.

To see a C++ Blob Storage sample, continue to:

[Azure Blob Storage SDK v12 for C++ sample](#)

# Quickstart: Upload, download, and list blobs using Go

8/22/2022 • 6 minutes to read • [Edit Online](#)

In this quickstart, you learn how to use the Go programming language to upload, download, and list block blobs in a container in Azure Blob storage.

## Prerequisites

To access Azure Storage, you'll need an Azure subscription. If you don't already have a subscription, create a [free account](#) before you begin.

All access to Azure Storage takes place through a storage account. For this quickstart, create a storage account using the [Azure portal](#), Azure PowerShell, or Azure CLI. For help creating a storage account, see [Create a storage account](#).

Make sure you have the following more prerequisites installed:

- [Go 1.17 or above](#)
- [Azure Storage Blob SDK for Go](#), using the following command:

```
go get -u github.com/Azure/azure-sdk-for-go/sdk/storage/azblob
```

### NOTE

Make sure that you capitalize `Azure` in the URL to avoid case-related import problems when working with the SDK. Also capitalize `Azure` in your import statements.

## Download the sample application

The [sample application](#) used in this quickstart is a basic Go application.

Use [git](#) to download a copy of the application to your development environment.

```
git clone https://github.com/Azure-Samples/storage-blobs-go-quickstart
```

This command clones the repository to your local git folder. To open the Go sample for Blob storage, look for `storage-quickstart.go` file.

## Sign in with Azure CLI

To support local development, the Azure Identity credential type `DefaultAzureCredential` authenticates users signed into the Azure CLI.

Run the following command to sign into the Azure CLI:

```
az login
```

Azure CLI authentication isn't recommended for applications running in Azure.

To learn more about different authentication methods, check out [Azure authentication with the Azure SDK for Go](#).

## Assign RBAC permissions to the storage account

Azure storage accounts require explicit permissions to perform read and write operations. In order to use the storage account, you must assign permissions to the account. To do that you'll need to assign an appropriate RBAC role to your account. To get the `<objectID>` of the currently signed in user, run

```
az ad signed-in-user show --query objectId
```

Run the following AzureCli command to assign the storage account permissions:

```
az role assignment create --assignee "<ObjectID>" --role "Storage Blob Data Contributor" --scope "<StorageAccountResourceID>"
```

Learn more about Azure's built-in RBAC roles, check out [Built-in roles](#).

Note: Azure Cli has built in helper fucntions that retrieve the storage access keys when permissions are not detected. That functionally does not transfer to the DefaultAzureCredential, which is the reason for assiging RBAC roles to your account.

## Run the sample

This sample creates an Azure storage container, uploads a blob, lists the blobs in the container, then downloads the blob data into a buffer.

Before you run the sample, open the `storage-quickstart.go` file. Replace `<StorageAccountName>` with the name of your Azure storage account.

Then run the application with the `go run` command:

```
go run storage-quickstart.go
```

The following output is an example of the output returned when running the application:

```
Azure Blob storage quick start sample
Creating a container named quickstart-4052363832531531139
Creating a dummy file to test the upload and download
Listing the blobs in the container:
blob-8721479556813186518

hello world this is a blob

Press enter key to delete the blob fils, example container, and exit the application.

Cleaning up.
Deleting the blob blob-8721479556813186518
Deleting the blob quickstart-4052363832531531139
```

When you press the key to continue, the sample program deletes the storage container and the files.

### TIP

You can also use a tool such as the [Azure Storage Explorer](#) to view the files in Blob storage. Azure Storage Explorer is a free cross-platform tool that allows you to access your storage account information.

# Understand the sample code

Next, we walk through the sample code so that you can understand how it works.

## Create ContainerURL and BlobURL objects

First, create the references to the ContainerURL and BlobURL objects used to access and manage Blob storage. These objects offer low-level APIs such as Create, Upload, and Download to issue REST APIs.

- Authenticate to Azure using the [DefaultAzureCredential](#).
- Use [NewServiceClient](#) struct to store your credentials.
- Instantiate a new [ContainerURL](#), and a new [BlockBlobClient](#) object to run operations on container (Create) and blobs (Upload and Download).

Once you have the ContainerURL, you can instantiate the BlobURL object that points to a blob, and perform operations such as upload, download, and copy.

### IMPORTANT

Container names must be lowercase. See [Naming and Referencing Containers, Blobs, and Metadata](#) for more information about container and blob names.

In this section, you create a new container. The container is called `quickstartblobs-[random string]`.

```
url := "https://storageblobsgo.blob.core.windows.net/"
ctx := context.Background()

// Create a default Azure credential
credential, err := azidentity.NewDefaultAzureCredential(nil)
if err != nil {
 log.Fatal("Invalid credentials with error: " + err.Error())
}

serviceClient, err := azblob.NewServiceClient(url, credential, nil)
if err != nil {
 log.Fatal("Invalid credentials with error: " + err.Error())
}

// Create the container
containerName := fmt.Sprintf("quickstart-%s", randomString())
fmt.Printf("Creating a container named %s\n", containerName)

containerClient := serviceClient.NewContainerClient(containerName)
_, err = containerClient.Create(ctx, nil)

if err != nil {
 log.Fatal(err)
}
```

## Upload blobs to the container

Blob storage supports block blobs, append blobs, and page blobs. Block blobs are the most commonly used, and that is what is used in this quickstart.

The SDK offers [high-level APIs](#) that are built on top of the low-level REST APIs. As an example, [UploadBufferToBlockBlob](#) function uses StageBlock (PutBlock) operations to concurrently upload a file in chunks to optimize the throughput. If the file is less than 256 MB, it uses Upload (PutBlob) instead to complete the transfer in a single transaction.

The following example uploads the file to your container called `quickstartblob-[randomstring]`.

```

data := []byte("\nhello world this is a blob\n")
blobName := "quickstartblob" + "-" + randomString()

var blockOptions azblob.HighLevelUploadToBlockBlobOption

blobClient, err := azblob.NewBlockBlobClient(url+containerName+"/"+blobName, credential, nil)
if err != nil {
 log.Fatal(err)
}

// Upload to data to blob storage
_, err = blobClient.UploadBufferToBlockBlob(ctx, data, blockOptions)

if err != nil {
 log.Fatalf("Failure to upload to blob: %v", err)
}

```

## List the blobs in a container

Get a list of files in the container using the [ListBlobs](#) method on a [ContainerURL](#). Blob names are returned in lexicographic order. After getting a segment, process it, and then call ListBlobs again.

```

// List the blobs in the container
pager := containerClient.ListBlobsFlat(nil)

for pager.NextPage(ctx) {
 resp := pager.PageResponse()

 for _, v := range resp.ContainerListBlobFlatSegmentResult.Segment.BlobItems {
 fmt.Println(*v.Name)
 }
}

if err = pager.Err(); err != nil {
 log.Fatalf("Failure to list blobs: %v", err)
}

// Download the blob
get, err := blobClient.Download(ctx, nil)
if err != nil {
 log.Fatal(err)
}

```

## Download the blob

Download blobs using the [Download](#) low-level function on a [BlobURL](#) will return a [DownloadResponse](#) struct. Run the function [Body](#) on the struct to get a [RetryReader](#) stream for reading data. If a connection fails while reading, it will make other requests to re-establish a connection and continue reading. Specifying a [RetryReaderOption](#)'s with [MaxRetryRequests](#) set to 0 (the default), returns the original response body and no retries will be performed. Alternatively, use the high-level APIs [DownloadBlobToBuffer](#) or [DownloadBlobToFile](#) to simplify your code.

The following code downloads the blob using the [Download](#) function. The contents of the blob is written into a buffer and shown on the console.

```
// Download the blob
get, err := blobClient.Download(ctx, nil)
if err != nil {
 log.Fatal(err)
}

downloadedData := &bytes.Buffer{}
reader := get.Body(azblob.RetryReaderOptions{})
_, err = downloadedData.ReadFrom(reader)
if err != nil {
 log.Fatal(err)
}
err = reader.Close()
if err != nil {
 log.Fatal(err)
}

fmt.Println(downloadedData.String())
```

## Clean up resources

If you no longer need the blobs uploaded in this quickstart, you can delete the entire container using the **Delete** method.

```
// Cleaning up the quick start by deleting the blob and container
fmt.Printf("Press enter key to delete the blob files, example container, and exit the application.\n")
bufio.NewReader(os.Stdin).ReadBytes('\n')
fmt.Printf("Cleaning up.\n")

// Delete the blob
fmt.Printf("Deleting the blob " + blobName + "\n")

_, err = blobClient.Delete(ctx, nil)
if err != nil {
 log.Fatalf("Failure: %+v", err)
}

// Delete the container
fmt.Printf("Deleting the blob " + containerName + "\n")
_, err = containerClient.Delete(ctx, nil)

if err != nil {
 log.Fatalf("Failure: %+v", err)
}
```

## Resources for developing Go applications with blobs

See these other resources for Go development with Blob storage:

- View and install the [Go client library source code](#) for Azure Storage on GitHub.
- Explore [Blob storage samples](#) written using the Go client library.

## Next steps

In this quickstart, you learned how to transfer files between a local disk and Azure blob storage using Go. For more information about the Azure Storage Blob SDK, view the [Source Code](#) and [API Reference](#).

# Transfer objects to/from Azure Blob storage using PHP

8/22/2022 • 6 minutes to read • [Edit Online](#)

In this quickstart, you learn how to use PHP to upload, download, and list block blobs in a container in Azure Blob storage.

## Prerequisites

To access Azure Storage, you'll need an Azure subscription. If you don't already have a subscription, create a [free account](#) before you begin.

All access to Azure Storage takes place through a storage account. For this quickstart, create a storage account using the [Azure portal](#), Azure PowerShell, or Azure CLI. For help creating a storage account, see [Create a storage account](#).

Make sure you have the following additional prerequisites installed:

- [PHP](#)
- [Azure Storage SDK for PHP](#)

## Download the sample application

The [sample application](#) used in this quickstart is a basic PHP application.

Use [git](#) to download a copy of the application to your development environment.

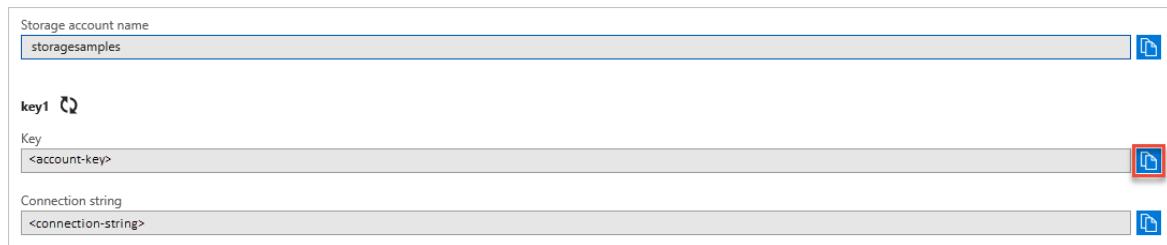
```
git clone https://github.com/Azure-Samples/storage-blobs-php-quickstart.git
```

This command clones the repository to your local git folder. To open the PHP sample application, look for the storage-blobs-php-quickstart folder, and open the phpqs.php file.

## Copy your credentials from the Azure portal

The sample application needs to authorize access to your storage account. Provide your storage account credentials to the application in the form of a connection string. To view your storage account credentials:

1. In to the [Azure portal](#) go to your storage account.
2. In the **Settings** section of the storage account overview, select **Access keys** to display your account access keys and connection string.
3. Note the name of your storage account, which you'll need for authorization.
4. Find the **Key** value under **key1**, and select **Copy** to copy the account key.



## Configure your storage connection string

In the application, you must provide your storage account name and account key to create the `BlobRestProxy` instance for your application. It is recommended to store these identifiers within an environment variable on the local machine running the application. Use one of the following examples depending on your Operating System to create the environment variable. Replace the `youraccountname` and `youraccountkey` values with your account name and key.

- [Linux](#)
- [Windows](#)

```
export ACCOUNT_NAME=<youraccountname>
export ACCOUNT_KEY=<youraccountkey>
```

## Configure your environment

Take the folder from your local git folder and place it in a directory served by your PHP server. Then, open a command prompt scoped to that same directory and enter: `php composer.phar install`

## Run the sample

This sample creates a test file in the '.' folder. The sample program uploads the test file to Blob storage, lists the blobs in the container, and downloads the file with a new name.

Run the sample. The following output is an example of the output returned when running the application:

```
Uploading BlockBlob: HelloWorld.txt
These are the blobs present in the container: HelloWorld.txt:
https://myexamplesacct.blob.core.windows.net/blockblobs1eqvxd>HelloWorld.txt

This is the content of the blob uploaded: Hello Azure!
```

When you press the button displayed, the sample program deletes the storage container and the files. Before you continue, check your server's folder for the two files. You can open them and see they are identical.

You can also use a tool such as the [Azure Storage Explorer](#) to view the files in Blob storage. Azure Storage Explorer is a free cross-platform tool that allows you to access your storage account information.

After you've verified the files, hit any key to finish the demo and delete the test files. Now that you know what the sample does, open the example.rb file to look at the code.

## Understand the sample code

Next, we walk through the sample code so that you can understand how it works.

### Get references to the storage objects

The first thing to do is create the references to the objects used to access and manage Blob storage. These

objects build on each other, and each is used by the next one in the list.

- Create an instance of the Azure storage `BlobRestProxy` object to set up connection credentials.
- Create the `BlobService` object that points to the Blob service in your storage account.
- Create the `Container` object, which represents the container you are accessing. Containers are used to organize your blobs like you use folders on your computer to organize your files.

Once you have the `blobClient` container object, you can create the `Block` blob object that points to the specific blob in which you are interested. Then you can perform operations such as upload, download, and copy.

#### IMPORTANT

Container names must be lowercase. See [Naming and Referencing Containers, Blobs, and Metadata](#) for more information about container and blob names.

In this section, you set up an instance of Azure storage client, instantiate the blob service object, create a new container, and set permissions on the container so the blobs are public. The container is called `quickstartblobs`.

```
Setup a specific instance of an Azure::Storage::Client
$connectionString =
"DefaultEndpointsProtocol=https;AccountName=".getenv('account_name').";AccountKey=".getenv('account_key');

// Create blob client.
$blobClient = BlobRestProxy::createBlobService($connectionString);

Create the BlobService that represents the Blob service for the storage account
$createContainerOptions = new CreateContainerOptions();

$createContainerOptions->setPublicAccess(PublicAccessType::CONTAINER_AND_BLOBS);

// Set container metadata.
$createContainerOptions->addMetaData("key1", "value1");
$createContainerOptions->addMetaData("key2", "value2");

$containerName = "blockblobs".generateRandomString();

try {
 // Create container.
 $blobClient->createContainer($containerName, $createContainerOptions);
```

#### Upload blobs to the container

Blob storage supports block blobs, append blobs, and page blobs. Block blobs are the most commonly used, and that is what is used in this quickstart.

To upload a file to a blob, get the full path of the file by joining the directory name and the file name on your local drive. You can then upload the file to the specified path using the `createBlockBlob()` method.

The sample code takes a local file and uploads it to Azure. The file is stored as `myfile` and the name of the blob as `fileToUpload` in the code. The following example uploads the file to your container called `quickstartblobs`.

```

$myfile = fopen("HelloWorld.txt", "w") or die("Unable to open file!");
fclose($myfile);

Upload file as a block blob
echo "Uploading BlockBlob: ".PHP_EOL;
echo $fileToUpload;
echo "
";

$content = fopen($fileToUpload, "r");

//Upload blob
$blobClient->createBlockBlob($containerName, $fileToUpload, $content);

```

To perform a partial update of the content of a block blob, use the `createBlockList()` method. Block blobs can be as large as 4.7 TB, and can be anything from Excel spreadsheets to large video files. Page blobs are primarily used for the VHD files used to back IaaS VMs. Append blobs are used for logging, such as when you want to write to a file and then keep adding more information. Append blob should be used in a single writer model. Most objects stored in Blob storage are block blobs.

### List the blobs in a container

You can get a list of files in the container using the `listBlobs()` method. The following code retrieves the list of blobs, then loops through them, showing the names of the blobs found in a container.

```

$listBlobsOptions = new ListBlobsOptions();
$listBlobsOptions->setPrefix("HelloWorld");

echo "These are the blobs present in the container: ";

do{
 $result = $blobClient->listBlobs($containerName, $listBlobsOptions);
 foreach ($result->getBlobs() as $blob)
 {
 echo $blob->getName()." ". $blob->getUrl()."
";
 }

 $listBlobsOptions->setContinuationToken($result->getContinuationToken());
} while($result->getContinuationToken());

```

### Get the content of your blobs

Get the contents of your blobs using the `getBlob()` method. The following code displays the contents of the blob uploaded in a previous section.

```

$blob = $blobClient->getBlob($containerName, $fileToUpload);
fpassthru($blob->getContentStream());

```

### Clean up resources

If you no longer need the blobs uploaded in this quickstart, you can delete the entire container using the `deleteContainer()` method. If the files created are no longer needed, you use the `deleteBlob()` method to delete the files.

```
// Delete blob.
echo "Deleting Blob".PHP_EOL;
echo $fileToUpload;
echo "
";
$blobClient->deleteBlob($_GET["containerName"], $fileToUpload);

// Delete container.
echo "Deleting Container".PHP_EOL;
echo $_GET["containerName"].PHP_EOL;
echo "
";
$blobClient->deleteContainer($_GET["containerName"]);

//Deleting local file
echo "Deleting file".PHP_EOL;
echo "
";
unlink($fileToUpload);
```

## Resources for developing PHP applications with blobs

See these additional resources for PHP development with Blob storage:

- View, download, and install the [PHP client library source code](#) for Azure Storage on GitHub.
- Explore [Blob storage samples](#) written using the PHP client library.

## Next steps

In this quickstart, you learned how to transfer files between a local disk and Azure blob storage using PHP. To learn more about working with PHP, continue to our PHP Developer center.

[PHP Developer Center](#)

For more information about the Storage Explorer and Blobs, see [Manage Azure Blob storage resources with Storage Explorer](#).

# Quickstart: Azure Blob Storage client library for Ruby

8/22/2022 • 5 minutes to read • [Edit Online](#)

Learn how to use Ruby to create, download, and list blobs in a container in Microsoft Azure Blob Storage.

## Prerequisites

To access Azure Storage, you'll need an Azure subscription. If you don't already have a subscription, create a [free account](#) before you begin.

All access to Azure Storage takes place through a storage account. For this quickstart, create a storage account using the [Azure portal](#), Azure PowerShell, or Azure CLI. For help creating a storage account, see [Create a storage account](#).

Make sure you have the following additional prerequisites installed:

- [Ruby](#)
- [Azure Storage library for Ruby](#), using the [RubyGem package](#):

```
gem install azure-storage-blob
```

## Download the sample application

The [sample application](#) used in this quickstart is a basic Ruby application.

Use [Git](#) to download a copy of the application to your development environment. This command clones the repository to your local machine:

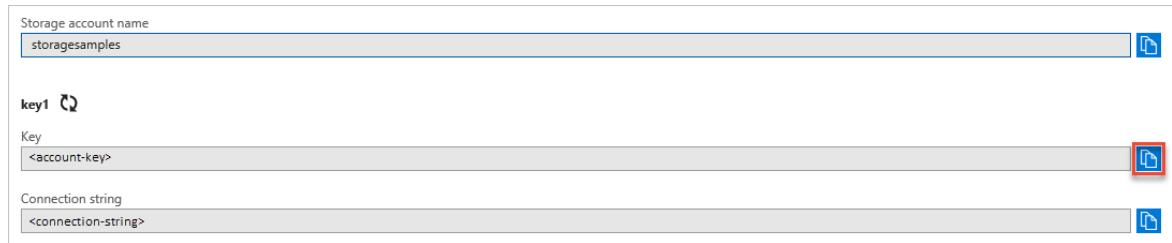
```
git clone https://github.com/Azure-Samples/storage-blobs-ruby-quickstart.git
```

Navigate to the *storage-blobs-ruby-quickstart* folder, and open the *example.rb* file in your code editor.

## Copy your credentials from the Azure portal

The sample application needs to authorize access to your storage account. Provide your storage account credentials to the application in the form of a connection string. To view your storage account credentials:

1. In the [Azure portal](#) go to your storage account.
2. In the **Settings** section of the storage account overview, select **Access keys** to display your account access keys and connection string.
3. Note the name of your storage account, which you'll need for authorization.
4. Find the **Key** value under **key1**, and select **Copy** to copy the account key.



## Configure your storage connection string

Provide your storage account name and account key to create a [BlobService](#) instance for your application.

The following code in the *example.rb* file instantiates a new [BlobService](#) object. Replace the *accountname* and *accountkey* values with your account name and key.

```
Create a BlobService object
account_name = "accountname"
account_key = "accountkey"

blob_client = Azure::Storage::Blob::BlobService.create(
 storage_account_name: account_name,
 storage_access_key: account_key
)
```

## Run the sample

The sample creates a container in Blob Storage, creates a new blob in the container, lists the blobs in the container, and downloads the blob to a local file.

Run the sample. Here is an example of the output from running the application:

```
C:\azure-samples\storage-blobs-ruby-quickstart> ruby example.rb

Creating a container: quickstartblobs18cd9ec0-f4ac-4688-a979-75c31a70503e

Creating blob: QuickStart_6f8f29a8-879a-41fb-9db2-0b8595180728.txt

List blobs in the container following continuation token
 Blob name: QuickStart_6f8f29a8-879a-41fb-9db2-0b8595180728.txt

Downloading blob to C:/Users/azureuser/Documents/QuickStart_6f8f29a8-879a-41fb-9db2-0b8595180728.txt

Paused, press the Enter key to delete resources created by the sample and exit the application
```

When you press Enter to continue, the sample program deletes the storage container and the local file. Before you continue, check your *Documents* folder for the downloaded file.

You can also use [Azure Storage Explorer](#) to view the files in your storage account. Azure Storage Explorer is a free cross-platform tool that allows you to access your storage account information.

After you've verified the files, press the Enter key to delete the test files and end the demo. Open the *example.rb* file to look at the code.

## Understand the sample code

Next, we walk through the sample code so you can understand how it works.

### Get references to the storage objects

The first thing to do is create instances of the objects used to access and manage Blob Storage. These objects build on each other. Each is used by the next one in the list.

- Create an instance of the Azure storage [BlobService](#) object to set up connection credentials.
- Create the [Container](#) object, which represents the container you're accessing. Containers are used to organize your blobs like you use folders on your computer to organize your files.

Once you have the container object, you can create a [Block](#) blob object that points to a specific blob in which you're interested. Use the [Block](#) object to create, download, and copy blobs.

#### IMPORTANT

Container names must be lowercase. For more information about container and blob names, see [Naming and Referencing Containers, Blobs, and Metadata](#).

The following example code:

- Creates a new container
- Sets permissions on the container so the blobs are public. The container is called *quickstartblobs* with a unique ID appended.

```
Create a container
container_name = "quickstartblobs" + SecureRandom.uuid
puts "\nCreating a container: " + container_name
container = blob_client.create_container(container_name)

Set the permission so the blobs are public
blob_client.set_container_acl(container_name, "container")
```

#### Create a blob in the container

Blob Storage supports block blobs, append blobs, and page blobs. To create a blob, call the [create\\_block\\_blob](#) method passing in the data for the blob.

The following example creates a blob called *QuickStart\_* with a unique ID and a *.txt* file extension in the container created earlier.

```
Create a new block blob containing 'Hello, World!'
blob_name = "QuickStart_" + SecureRandom.uuid + ".txt"
blob_data = "Hello, World!"
puts "\nCreating blob: " + blob_name
blob_client.create_block_blob(container.name, blob_name, blob_data)
```

Block blobs can be as large as 4.7 TB, and can be anything from spreadsheets to large video files. Page blobs are primarily used for the VHD files that back IaaS virtual machines. Append blobs are commonly used for logging, such as when you want to write to a file and then keep adding more information.

#### List the blobs in a container

Get a list of files in the container using the [list\\_blobs](#) method. The following code retrieves the list of blobs, then displays their names.

```

List the blobs in the container
puts "\nList blobs in the container following continuation token"
nextMarker = nil
loop do
 blobs = blob_client.list_blobs(container_name, { marker: nextMarker })
 blobs.each do |blob|
 puts "\tBlob name: #{blob.name}"
 end
 nextMarker = blobs.continuation_token
 break unless nextMarker && !nextMarker.empty?
end

```

## Download a blob

Download a blob to your local disk using the [get\\_blob](#) method. The following code downloads the blob created in a previous section.

```

Download the blob

Set the path to the local folder for downloading
if(is_windows)
 local_path = File.expand_path("~/Documents")
else
 local_path = File.expand_path("~/")
end

Create the full path to the downloaded file
full_path_to_file = File.join(local_path, blob_name)

puts "\nDownloading blob to " + full_path_to_file
blob, content = blob_client.get_blob(container_name, blob_name)
File.open(full_path_to_file, "wb") {|f| f.write(content)}

```

## Clean up resources

If a blob is no longer needed, use [delete\\_blob](#) to remove it. Delete an entire container using the [delete\\_container](#) method. Deleting a container also deletes any blobs stored in the container.

```

Clean up resources, including the container and the downloaded file
blob_client.delete_container(container_name)
File.delete(full_path_to_file)

```

## Resources for developing Ruby applications with blobs

See these additional resources for Ruby development:

- View and download the [Ruby client library source code](#) for Azure Storage on GitHub.
- Explore [Azure samples](#) written using the Ruby client library.
- [Sample: Getting Started with Azure Storage in Ruby](#)

## Next steps

In this quickstart, you learned how to transfer files between Azure Blob Storage and a local disk by using Ruby. To learn more about working with Blob Storage, continue to the Storage account overview.

### [Storage account overview](#)

For more information about the Storage Explorer and Blobs, see [Manage Azure Blob Storage resources with Storage Explorer](#).

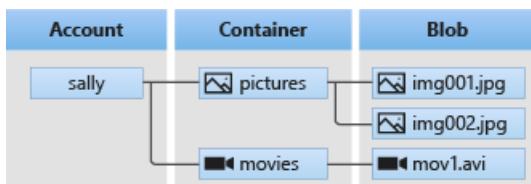
# Quickstart: Use Azure Blob Storage client library v12 with Xamarin

8/22/2022 • 5 minutes to read • [Edit Online](#)

This quickstart gets you started using Xamarin with the Azure Blob Storage client library v12. The Xamarin mobile development framework creates C# apps for iOS, Android, and UWP from one .NET codebase.

Blob Storage is optimized for storing massive amounts of unstructured data, like text or binary data, that doesn't fit a particular data model or definition. Blob Storage has three types of resources: a storage account, containers in the storage account, and blobs in the containers.

The following diagram shows the relationship between these types of resources:



You can use the following .NET classes to interact with Blob Storage resources:

- [BlobServiceClient](#) manipulates Storage resources and blob containers.
- [BlobContainerClient](#) manipulates Storage containers and their blobs.
- [BlobClient](#) manipulates Storage blobs.
- [BlobDownloadInfo](#) represents the properties and content returned from downloading a blob.

In this quickstart, you use Xamarin with the Azure Blob Storage client library v12 to:

- [Create a container](#)
- [Upload blobs to a container](#)
- [List the blobs in a container](#)
- [Download blobs](#)
- [Delete a container](#)

## Prerequisites

- Azure subscription. [Create one for free](#).
- Azure Storage account. [Create a storage account](#).
- Visual Studio with the [Mobile Development for .NET](#) workload installed, or [Visual Studio for Mac](#)

## Visual Studio setup

This section walks through preparing a Visual Studio Xamarin project to work with the Azure Blob Storage client library v12.

1. In Visual Studio, create a Blank Forms App named *BlobQuickstartV12*.
2. In Visual Studio **Solution Explorer**, right-click the solution and select **Manage NuGet Packages for Solution**.
3. Search for **Azure.Storage.Blobs**, and install the latest stable version into all projects in your solution.

4. In Solution Explorer, from the BlobQuickstartV12 directory, open the *MainPage.xaml* file for editing.

5. In the code editor, replace everything between the `<ContentPage></ContentPage>` elements with the following code:

```
<StackLayout HorizontalOptions="Center" VerticalOptions="Center">

 <Button x:Name="uploadButton" Text="Upload Blob" Clicked="Upload_Clicked" IsEnabled="False"/>
 <Button x:Name="listButton" Text="List Blobs" Clicked="List_Clicked" IsEnabled="False" />
 <Button x:Name="downloadButton" Text="Download Blob" Clicked="Download_Clicked"
IsEnabled="False" />
 <Button x:Name="deleteButton" Text="Delete Container" Clicked="Delete_Clicked" IsEnabled="False"
/>

 <Label Text="" x:Name="resultsLabel" HorizontalTextAlignment="Center" Margin="0,20,0,0"
TextColor="Red" />

</StackLayout>
```

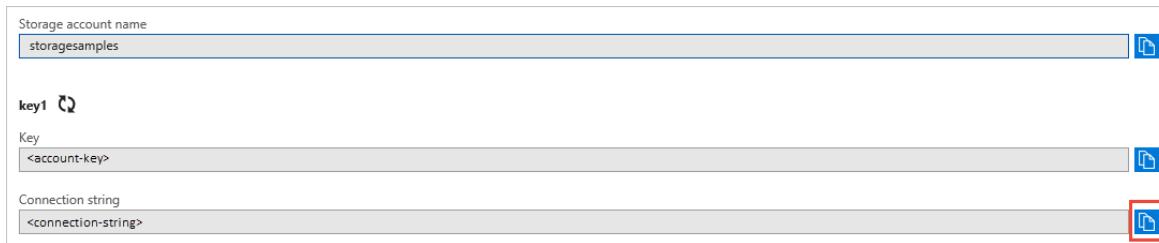
## Azure Storage connection

To authorize requests to Azure Storage, you need to add your storage account credentials to your application as a connection string.

### Copy your credentials from the Azure portal

When the sample application makes a request to Azure Storage, it must be authorized. To authorize a request, add your storage account credentials to the application as a connection string. View your storage account credentials by following these steps:

1. Sign in to the [Azure portal](#).
2. Locate your storage account.
3. In the **Settings** section of the storage account overview, select **Access keys**. Here, you can view your account access keys and the complete connection string for each key.
4. Find the **Connection string** value under **key1**, and select the **Copy** button to copy the connection string. You will add the connection string value to an environment variable in the next step.



### Configure your storage connection string

After you have copied your connection string, set it to a class level variable in your *MainPage.xaml.cs* file. Open up *MainPage.xaml.cs* and find the `storageConnectionString` variable. Replace `<yourconnectionstring>` with your actual connection string.

Here's the code:

```
string storageConnectionString = "<yourconnectionstring>";
```

## Code examples

The following example code snippets show you how to use the Blob Storage client library for .NET in a Xamarin.Forms app.

### Create class level variables

The following code declares several class-level variables that the samples use to communicate with Blob Storage. Add these lines to *MainPage.xaml.cs*, immediately after the storage account connection string you just added.

```
string fileName = $"{Guid.NewGuid()}-temp.txt";

BlobServiceClient client;
BlobContainerClient containerClient;
BlobClient blobClient;
```

### Create a container

The code creates an instance of the [BlobServiceClient](#) class, then calls the [CreateBlobContainerAsync](#) method to create the container in your storage account.

The code appends a GUID value to the container name to ensure that it's unique. For more information about naming containers and blobs, see [Name and reference containers, blobs, and metadata](#).

Add the following code to the *MainPage.xaml.cs* file:

```
protected async override void OnAppearing()
{
 string containerName = $"quickstartblobs{Guid.NewGuid()}";

 client = new BlobServiceClient(storageConnectionString);
 containerClient = await client.CreateBlobContainerAsync(containerName);

 resultsLabel.Text = "Container Created\n";

 blobClient = containerClient.GetBlobClient(fileName);

 uploadButton.IsEnabled = true;
}
```

### Upload blobs to a container

The following code snippet:

1. Creates a `MemoryStream` of the text.
2. Uploads the text to a blob by calling the `UploadAsync` function of the `BlobContainerClient` class. The code passes in both the filename and the `MemoryStream` of text. This method creates the blob if it doesn't already exist, and overwrites it if it does.

Add the following code to the *MainPage.xaml.cs* file:

```
async void Upload_Clicked(object sender, EventArgs e)
{
 using MemoryStream memoryStream = new MemoryStream(Encoding.UTF8.GetBytes("Hello World!"));

 await containerClient.UploadBlobAsync(fileName, memoryStream);

 resultsLabel.Text += "Blob Uploaded\n";

 uploadButton.IsEnabled = false;
 listButton.IsEnabled = true;
}
```

## List the blobs in a container

This code lists the blobs in the container by calling the [GetBlobsAsync](#) method. You added only one blob to the container, so the listing operation returns just one blob.

Add this code to the *MainPage.xaml.cs* file:

```
async void List_Clicked(object sender, EventArgs e)
{
 await foreach (BlobItem blobItem in containerClient.GetBlobsAsync())
 {
 resultsLabel.Text += blobItem.Name + "\n";
 }

 listButton.IsEnabled = false;
 downloadButton.IsEnabled = true;
}
```

## Download blobs

Download the blob you previously created by calling the [DownloadToAsync](#) method. The example code copies the `Stream` representation of the blob into a `MemoryStream` and then into a `StreamReader` to display the text.

Add this code to the *MainPage.xaml.cs* file:

```
async void Download_Clicked(object sender, EventArgs e)
{
 BlobDownloadInfo downloadInfo = await blobClient.DownloadAsync();

 using MemoryStream memoryStream = new MemoryStream();

 await downloadInfo.Content.CopyToAsync(memoryStream);
 memoryStream.Position = 0;

 using StreamReader streamReader = new StreamReader(memoryStream);

 resultsLabel.Text += "Blob Contents: \n";
 resultsLabel.Text += await streamReader.ReadToEndAsync();
 resultsLabel.Text += "\n";

 downloadButton.IsEnabled = false;
 deleteButton.IsEnabled = true;
}
```

## Delete a container

The following code deletes the container and its blobs, by using [DeleteAsync](#).

The app first prompts you to confirm the blob and container deletion. You can then verify that the resources were created correctly, before you delete them.

Add this code to the *MainPage.xaml.cs* file:

```

async void Delete_Clicked(object sender, EventArgs e)
{
 var deleteContainer = await Application.Current.MainPage.DisplayAlert("Delete Container",
 "You're about to delete the container. Proceed?", "OK", "Cancel");

 if (deleteContainer == false)
 return;

 await containerClient.DeleteAsync();

 resultsLabel.Text += "Container Deleted";

 deleteButton.IsEnabled = false;
}

```

## Run the code

After you add all the code, to run the app on Windows press F5. To run the app on Mac, press Cmd+Enter. When the app starts, it first creates the container. You can then select the buttons to upload, list, and download the blobs, and delete the container.

The app writes to the screen after every operation, with output similar to the following example:

```

Container Created
Blob Uploaded
98d9a472-8e98-4978-ba4f-081d69d2e6f8-temp.txt
Blob Contents:
Hello World!
Container Deleted

```

Before you begin the clean-up process, verify that the output of the blob's contents match the blob that you uploaded. After you verify the values, confirm the container deletion to finish the quickstart.

## Next steps

In this quickstart, you learned how to use Xamarin to create and delete containers, and upload, download, and list blobs, with the Azure Blob Storage client library v12.

To see Blob storage sample apps, continue to:

### [Azure Blob Storage SDK v12 Xamarin sample](#)

- For tutorials, samples, quick starts and other documentation, visit [Azure for mobile developers](#).
- To learn more about Xamarin, see [Get started with Xamarin](#).

Azure.Storage.Blobs reference links:

- [API reference documentation](#)
- [Client library source code](#)
- [NuGet package](#)

# Tutorial: Upload and analyze a file with Azure Functions and Blob Storage

8/22/2022 • 13 minutes to read • [Edit Online](#)

In this tutorial, you'll learn how to upload an image to Azure Blob Storage and process it using Azure Functions and Computer Vision. You'll also learn how to implement Azure Function triggers and bindings as part of this process. Together, these services will analyze an uploaded image that contains text, extract the text out of it, and then store the text in a database row for later analysis or other purposes.

Azure Blob Storage is Microsoft's massively scalable object storage solution for the cloud. Blob Storage is designed for storing images and documents, streaming media files, managing backup and archive data, and much more. You can read more about Blob Storage on the [overview page](#).

Azure Functions is a serverless computer solution that allows you to write and run small blocks of code as highly scalable, serverless, event driven functions. You can read more about Azure Functions on the [overview page](#).

In this tutorial, you will learn how to:

- Upload images and files to Blob Storage
- Use an Azure Function event trigger to process data uploaded to Blob Storage
- Use Cognitive Services to analyze an image
- Write data to Table Storage using Azure Function output bindings

## Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- [Visual Studio 2022](#) installed.

## Create the storage account and container

The first step is to create the storage account that will hold the uploaded blob data, which in this scenario will be images that contain text. A storage account offers several different services, but this tutorial utilizes Blob Storage and Table Storage.

- [Azure portal](#)
- [Azure CLI](#)

Sign in to the [Azure portal](#).

1. In the search bar at the top of the portal, search for *Storage* and select the result labeled **Storage accounts**.
2. On the **Storage accounts** page, select **+ Create** in the top left.
3. On the **Create a storage account** page, enter the following values:
  - **Subscription:** Choose your desired subscription.
  - **Resource Group:** Select **Create new** and enter a name of `msdocs-storage-function`, and then choose **OK**.
  - **Storage account name:** Enter a value of `msdocsstoragefunction`. The Storage account name must be unique across Azure, so you may need to add numbers after the name, such as `msdocsstoragefunction123`.

- **Region:** Select the region that is closest to you.
- **Performance:** Choose Standard.
- **Redundancy:** Leave the default value selected.

Home > Storage accounts >

### Create a storage account

[Basics](#) [Advanced](#) [Networking](#) [Data protection](#) [Encryption](#) [Tags](#) [Review + create](#)

Azure Storage is a Microsoft-managed service providing cloud storage that is highly available, secure, durable, scalable, and redundant. Azure Storage includes Azure Blobs (objects), Azure Data Lake Storage Gen2, Azure Files, Azure Queues, and Azure Tables. The cost of your storage account depends on the usage and the options you choose below. [Learn more about Azure storage accounts](#)

**Project details**

Select the subscription in which to create the new storage account. Choose a new or existing resource group to organize and manage your storage account together with other resources.

Subscription: C&L Cross Service Content Team Testing

Resource group: (View) msdocs-storage-function [Create new](#)

**Instance details**

If you need to create a legacy storage account type, please click here.

Storage account name: msdocsstoragefunction

Region: (US) East US

Performance: Standard: Recommended for most scenarios (general-purpose v2 account)  Premium: Recommended for scenarios that require low latency.

Redundancy: Geo-redundant storage (GRS)  Make read access to data available in the event of regional unavailability.

4. Select **Review + Create** at the bottom and Azure will validate the information you entered. Once the settings are validated, choose **Create** and Azure will begin provisioning the storage account, which might take a moment.

## Create the container

1. After the storage account is provisioned, select **Go to Resource**. The next step is to create a storage container inside of the account to hold uploaded images for analysis.
2. On the navigation panel, choose **Containers**.
3. On the **Containers** page, select **+ Container** at the top. In the slide out panel, enter a **Name** of *imageanalysis*, and make sure the **Public access level** is set to **Blob (anonymous read access for blobs only)**. Then select **Create**.

Home > msdocsstoragefunction\_1648483329772 > msdocsstoragefunction

**msdocsstoragefunction | Containers** [...](#)

Storage account

Search (Ctrl+I) [Container](#) [Change access level](#) [Restore containers](#) [Refresh](#) [...](#)

Overview [Activity log](#) [Tags](#) [Diagnose and solve problems](#) [Access Control \(IAM\)](#) [Data migration](#) [Events](#) [Storage browser \(preview\)](#)

Data storage

Containers [File shares](#) [Queues](#) [Tables](#)

**New container**

Name:

Public access level: [Blob \(anonymous read access for blobs only\)](#)

⚠️ Blobs within the container can be read by anonymous request, but container data is not available. Anonymous clients cannot enumerate the blobs within the container.

[Advanced](#)

You should see your new container appear in the list of containers.

## Retrieve the connection string

The last step is to retrieve our connection string for the storage account.

1. On the left navigation panel, select **Access Keys**.
2. On the **Access Keys page**, select **Show keys**. Copy the value of the **Connection String** under the **key1** section and paste this somewhere to use for later. You'll also want to make a note of the storage account name `msdocsstoragefunction` for later as well.

The screenshot shows the 'Access keys' page for a storage account named 'msdocsstoragefunction'. The left sidebar lists various storage management options. The 'Access keys' option is selected and highlighted with a red box. At the top, there's a search bar and a 'Show keys' button, which is also highlighted with a red box. Below it are buttons for 'Set rotation reminder' and 'Refresh'. A note about using keys securely is present. The main area displays the storage account name 'msdocsstoragefunction'. Under 'key1', the 'Last rotated' date is shown as '3/28/2022 (0 days ago)'. There's a 'Rotate key' button and a 'Key' input field containing a long, obscured string. Below this is a 'Connection string' input field, also containing a long, obscured string and highlighted with a red box. Under 'key2', similar information is provided, though it has not been rotated yet.

These values will be necessary when we need to connect our Azure Function to this storage account.

## Create the Computer Vision service

Next, create the Computer Vision service account that will process our uploaded files. Computer Vision is part of Azure Cognitive Services and offers a variety of features for extracting data out of images. You can learn more about Computer Vision on the [overview page](#).

- [Azure portal](#)
- [Azure CLI](#)

1. In the search bar at the top of the portal, search for *Computer* and select the result labeled **Computer vision**.
2. On the **Computer vision** page, select **+ Create**.
3. On the **Create Computer Vision** page, enter the following values:
  - **Subscription:** Choose your desired Subscription.
  - **Resource Group:** Use the `msdocs-storage-function` resource group you created earlier.
  - **Region:** Select the region that is closest to you.
  - **Name:** Enter in a name of `msdocscomputervision`.
  - **Pricing Tier:** Choose **Free** if it is available, otherwise choose **Standard S1**.
  - Check the **Responsible AI Notice** box if you agree to the terms

Basics Network Identity Tags Review + create

Boost content discoverability, accelerate text extraction, and create products that more people can use by embedding vision capabilities in your apps. Use visual data processing to label content (from objects to concepts), extract printed and handwritten text, recognize familiar subjects like brands and landmarks, and moderate content. No machine learning expertise is required.

Learn more

**Project Details**

Subscription \*

Resource group \*

**Instance Details**

Region

Name \*

Pricing tier \*

[View full pricing details](#)

**Responsible AI Notice**

Microsoft provides technical documentation regarding the appropriate operation applicable to this Cognitive Service that is made available by Microsoft. Customer acknowledges and agrees that they have reviewed this documentation and will use this service in accordance with it. This Cognitive Services is intended to process Customer Data that includes Biometric Data (as may be further described in product documentation) that Customer may incorporate into its own systems used for personal identification or other purposes. Customer acknowledges and agrees that it is responsible for complying with the Biometric Data obligations contained in the Online Services DPA.

[Online Services DPA](#)

[Responsible Use of AI documentation for Spatial Analysis](#)

By checking this box I certify that I have  reviewed and acknowledge the all the terms above.

4. Select **Review + Create** at the bottom. Azure will take a moment validate the information you entered.  
Once the settings are validated, choose **Create** and Azure will begin provisioning the Computer Vision service, which might take a moment.
5. When the operation has completed, select **Go to Resource**.

### Retrieve the keys

Next, we need to find the secret key and endpoint URL for the Computer Vision service to use in our Azure Function app.

1. On the **Computer Vision** overview page, select **Keys and Endpoint**.
2. On the **Keys and EndPoint** page, copy the **Key 1** value and the **EndPoint** values and paste them somewhere to use for later.

The screenshot shows the Azure portal interface for the 'msdocscomputervision' service. On the left, there's a sidebar with various management options like Overview, Activity log, Access control (IAM), Tags, and Diagnose and solve problems. Below that is a 'Resource Management' section with 'Keys and Endpoint' selected, which is highlighted with a red box. The main content area has a 'Show Keys' button. It displays two key fields: 'KEY 1' and 'KEY 2', both of which are also highlighted with red boxes. Below the keys is a 'Location/Region' dropdown set to 'eastus'. At the bottom is an 'Endpoint' field containing the URL 'https://msdocscomputervision.cognitiveservices.azure.com/'. There are also 'Regenerate Key1' and 'Regenerate Key2' buttons at the top of the keys section.

## Download and configure the sample project

The code for the Azure Function used in this tutorial can be found in [this GitHub repository](#). You can also clone the project using the command below.

```
git clone https://github.com/Azure-Samples/msdocs-storage-bind-function-service.git \
cd msdocs-storage-bind-function-service/dotnet
```

The sample project code accomplishes the following tasks:

- Retrieves environment variables to connect to the storage account and Computer Vision service
- Accepts the uploaded file as a blob parameter
- Analyzes the blob using the Computer Vision service
- Sends the analyzed image text to a new table row using output bindings

Once you have downloaded and opened the project, there are a few essential concepts to understand in the main `Run` method shown below. The Azure function utilizes Trigger and Output bindings, which are applied using attributes on the `Run` method signature.

The `Table` attribute uses two parameters. The first parameter specifies the name of the table to write the parsed image text value returned by the function. The second `connection` parameter pulls a Table Storage connection string from the environment variables so that our Azure function has access to it.

The `BlobTrigger` attribute is used to bind our function to the upload event in Blob Storage, and supplies that uploaded blob to the `Run` function. The blob trigger has two parameters of its own - one for the name of the blob container to monitor for uploads, and one for the connection string of our storage account again.

```

// Azure Function name and output Binding to Table Storage
[FunctionName("ProcessImageUpload")]
[return: Table("ImageText", Connection = "StorageConnection")]
// Trigger binding runs when an image is uploaded to the blob container below
public async Task<ImageContent> Run([BlobTrigger("imageanalysis/{name}",
 Connection = "StorageConnection")]Stream myBlob, string name, ILogger log)
{
 // Get connection configurations
 string subscriptionKey = Environment.GetEnvironmentVariable("ComputerVisionKey");
 string endpoint = Environment.GetEnvironmentVariable("ComputerVisionEndpoint");
 string imgUrl = $"https://{{ Environment.GetEnvironmentVariable("StorageAccountName") }}.blob.core.windows.net/imageanalysis/{{name}}";

 ComputerVisionClient client = new ComputerVisionClient(
 new ApiKeyServiceClientCredentials(subscriptionKey)) { Endpoint = endpoint };

 // Get the analyzed image contents
 var textContext = await AnalyzeImageContent(client, imgUrl);

 return new ImageContent {
 PartitionKey = "Images",
 RowKey = Guid.NewGuid().ToString(), Text = textContext
 };
}

public class ImageContent
{
 public string PartitionKey { get; set; }
 public string RowKey { get; set; }
 public string Text { get; set; }
}

```

This code also retrieves essential configuration values from environment variables, such as the storage account connection string and Computer Vision key. We'll add these environment variables to our Azure Function environment after it's deployed.

The `ProcessImage` function also utilizes a second method called `AnalyzeImage`, seen below. This code uses the URL Endpoint and Key of our Computer Vision account to make a request to that server to process our image. The request will return all of the text discovered in the image, which will then be written to Table Storage using the output binding on the `Run` method.

```

static async Task<string> ReadImageUrl(ComputerVisionClient client, string urlFile)
{
 // Analyze the file using Computer Vision Client
 var textHeaders = await client.ReadAsync(urlFile);
 string operationLocation = textHeaders.OperationLocation;
 Thread.Sleep(2000);

 // Complete code omitted for brevity, view in sample project

 return text.ToString();
}

```

## Running locally

If you'd like to run the project locally, you can populate the environment variables using the `local.settings.json` file. Inside of this file, fill in the placeholder values with the values you saved earlier when creating the Azure resources.

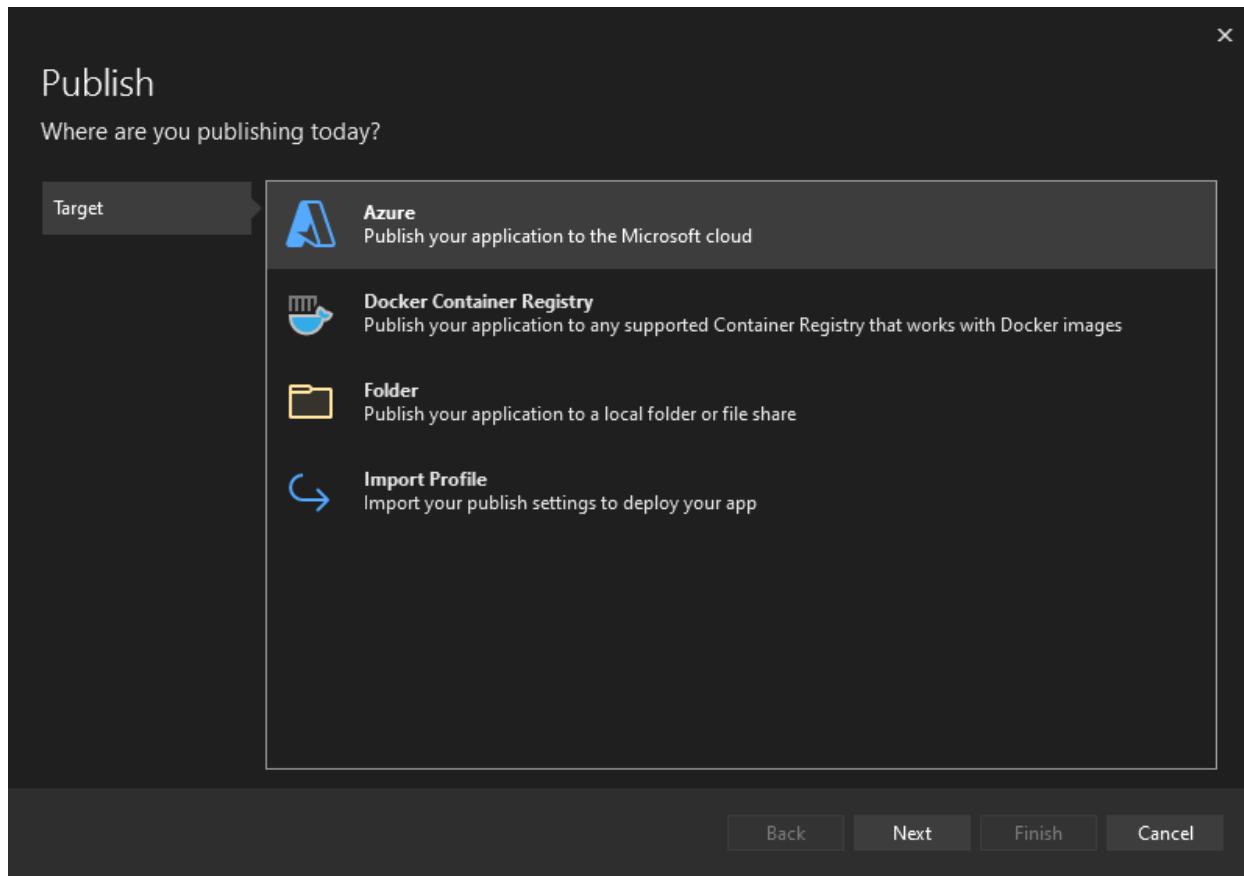
Although the Azure Function code will run locally, it will still connect to the live services out on Azure, rather than using any local emulators.

```
{
 "IsEncrypted": false,
 "Values": {
 "AzureWebJobsStorage": "UseDevelopmentStorage=true",
 "FUNCTIONS_WORKER_RUNTIME": "dotnet",
 "StorageConnection": "your-storage-account-connection-string",
 "StorageAccountName": "your-storage-account-name",
 "ComputerVisionKey": "your-computer-vision-key",
 "ComputerVisionEndPoint": "your-computer-vision-endpoint"
 }
}
```

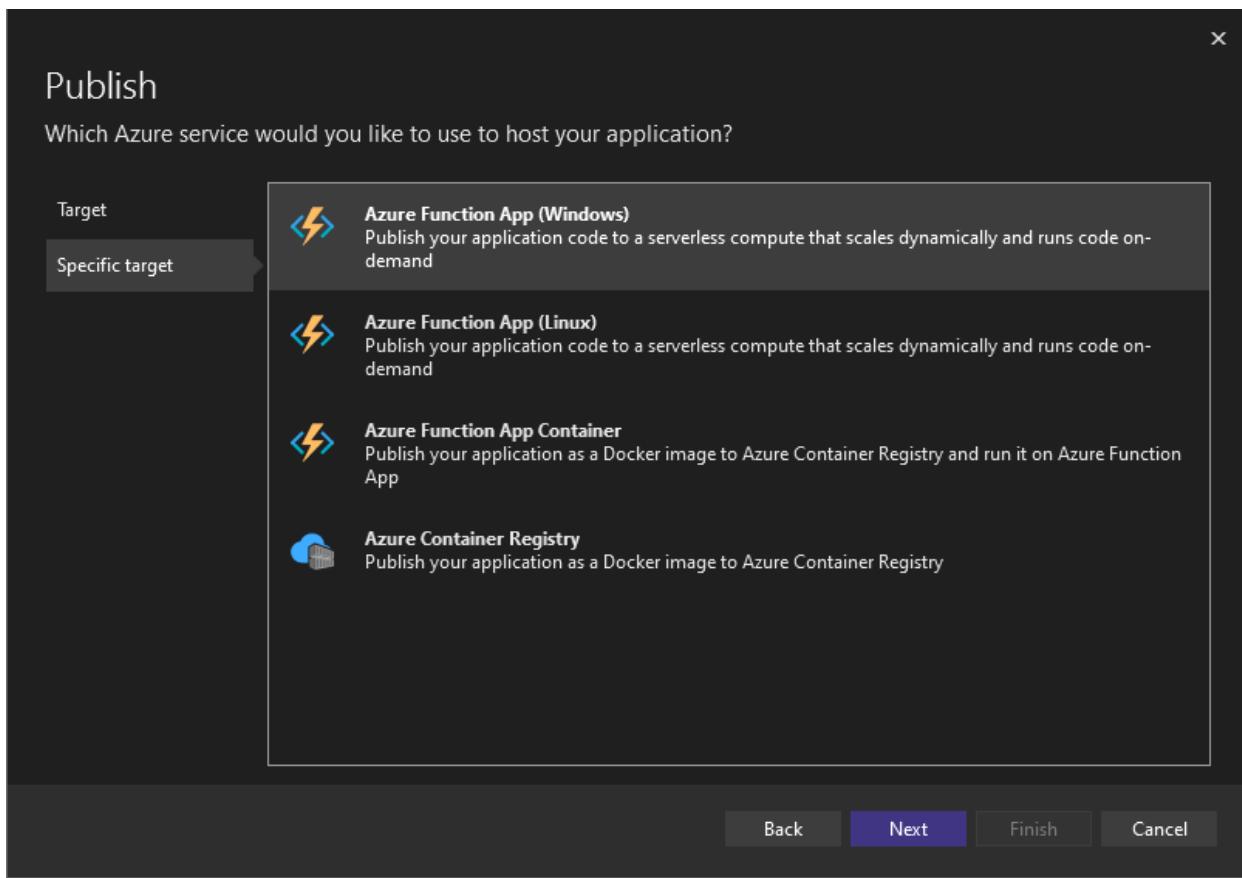
## Deploy the code to Azure Functions

You are now ready to deploy our application to Azure by using Visual Studio. You can also create the Azure Functions app in Azure at the same time as part of the deployment process.

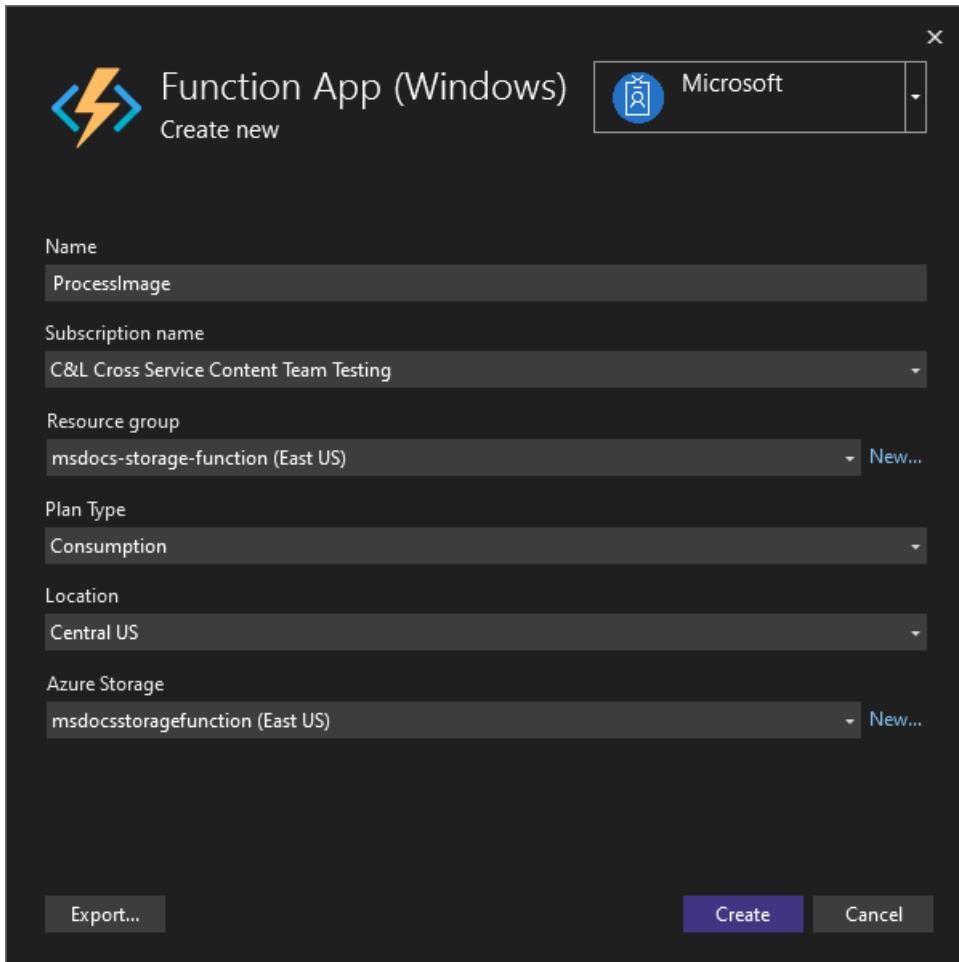
1. To begin, right select the **ProcessImage** project node and select **Publish**.
2. On the **Publish** dialog screen, select Azure and choose **Next**.



3. Select **Azure Function App (Windows)** or **Azure Function App (Linux)** on the next screen, and then choose **Next** again.



4. On the **Functions instance** step, make sure to choose the subscription you'd like to deploy to. Next, select the green + symbol on the right side of the dialog.
5. A new dialog will open. Enter the following values for your new Function App.
  - **Name:** Enter *msdocsprocessimage* or something similar.
  - **Subscription Name:** Choose whatever subscription you'd like to use.
  - **Resource Group:** Choose the `msdocs-storage-function` resource group you created earlier.
  - **Plan Type:** Select **Consumption**.
  - **Location:** Choose the region closest to you.
  - **Azure Storage:** Select the storage account you created earlier.



6. Once you have filled in all of those values, select **Create**. Visual Studio and Azure will begin provisioning the requested resources, which will take a few moments to complete.
7. Once the process has finished, select **Finish** to close out the dialog workflow.
8. The final step to deploy the Azure Function is to select **Publish** in the upper right of the screen. Publishing the function might also take a few moments to complete. Once it finishes, your application will be running on Azure.

## Connect the services

The Azure Function was deployed successfully, but it cannot connect to our storage account and Computer Vision services yet. The correct keys and connection strings must first be added to the configuration settings of the Azure Functions app.

1. At the top of the Azure portal, search for *function* and select **Function App** from the results.
2. On the **Function App** screen, select the Function App you created in Visual Studio.
3. On the **Function App** overview page, select **Configuration** on the left navigation. This will open up a page where we can manage various types of configuration settings for our app. For now, we are interested in **Application Settings** section.
4. The next step is to add settings for our storage account name and connection string, the Computer Vision secret key, and the Computer Vision endpoint.
5. On the **Application settings** tab, select **+ New application setting**. In the flyout that appears, enter the following values:
  - **Name:** Enter a value of *ComputerVisionKey*.
  - **Value:** Paste in the Computer Vision key you saved from earlier.

6. Click **OK** to add this setting to your app.

The screenshot shows the Azure portal's configuration page for an application. On the left, there are tabs for 'Application settings', 'Function runtime settings', and 'General'. The 'Application settings' tab is selected. Below it, a sub-section titled 'Application settings' explains that they are encrypted at rest and transmitted over a secure connection. It includes links to 'Learn more' and 'Show values'. A red box highlights the '+ New application setting' button. To its right are 'Show values' and 'Advanced' buttons, and a 'Filter application settings' input field. On the right, a modal window titled 'Add/Edit application setting' is open. It has fields for 'Name' (set to 'ComputerVisionKey') and 'Value' (containing a long string of asterisks). A checkbox for 'Deployment slot setting' is present but unchecked. A red box highlights the entire modal window.

7. Next, let's repeat this process for the endpoint of our Computer Vision service, using the following values:

- **Name:** Enter a value of *ComputerVisionEndpoint*.
- **Value:** Paste in the endpoint URL you saved from earlier.

8. Repeat this step again for the storage account connection, using the following values:

- **Name:** Enter a value of *StorageConnection*.
- **Value:** Paste in the connection string you saved from earlier.

9. Finally, repeat this process one more time for the storage account name, using the following values:

- **Name:** Enter a value of *StorageAccountName*.
- **Value:** Enter in the name of the storage account you created.

10. After you have added these application settings, make sure to select **Save** at the top of the configuration page. When the save completes, you can hit **Refresh** as well to make sure the settings are picked up.

All of the required environment variables to connect our Azure function to different services are now in place.

## Upload an image to Blob Storage

You are now ready to test out our application! You can upload a blob to the container, and then verify that the text in the image was saved to Table Storage.

1. First, at the top of the Azure portal, search for *Storage* and select **storage account**. On the **storage account** page, select the account you created earlier.
2. Next, select **Containers** on the left nav, and then navigate into the **ImageAnalysis** container you created earlier. From here you can upload a test image right inside the browser.

The screenshot shows the Azure Storage account 'msdocsstoragefunction' with the 'Containers' blade open. The left sidebar includes links like Overview, Activity log, Tags, Diagnose and solve problems, Access Control (IAM), Data migration, Events, and Storage browser (preview). Under Data storage, 'Containers' is selected and highlighted with a red box. The main area lists containers: '\$logs', 'azure-webjobs-hosts', 'azure-webjobs-secrets', and 'imageanalysis', which is also highlighted with a red box.

3. You can find a few sample images included in the **images** folder at the root of the downloadable sample project, or you can use one of your own.
4. At the top of the **ImageAnalysis** page, select **Upload**. In the flyout that opens, select the folder icon on the right to open up a file browser. Choose the image you'd like to upload, and then select **Upload**.

The screenshot shows the 'imageanalysis' container page in Azure Storage. The left sidebar includes links like Overview, Diagnose and solve problems, Access Control (IAM), Settings, Shared access tokens, Access policy, Properties, Metadata, and Editor (preview). The 'Upload' button is highlighted with a red box. To the right, an 'Upload blob' dialog box is open, also highlighted with a red box. It contains fields for 'Select a file' (with a browse icon) and 'Overwrite if files already exist' (checkbox). There is also an 'Advanced' section and a large 'Upload' button at the bottom.

5. The file should appear inside of your blob container. Next, you can verify that the upload triggered the Azure Function, and that the text in the image was analyzed and saved to Table Storage properly.
6. Using the breadcrumbs at the top of the page, navigate up one level in your storage account. Locate and select **Storage browser** on the left nav, and then select **Tables**.
7. An **ImageText** table should now be available. Click on the table to preview the data rows inside of it. You should see an entry for the processed image text of our upload. You can verify this using either the **Timestamp**, or by viewing the content of the **Text** column.

The screenshot shows the Azure Storage browser (preview) interface for the storage account 'msdocsstoragefunction'. The left sidebar has a 'Storage browser (preview)' section selected. The main area shows a table named 'ImageText' with one item listed. The table has columns for Name and Url, with the URL being 'https://msdocsstoragefunction.table.core.windows.net'.

Congratulations! You succeeded in processing an image that was uploaded to Blob Storage using Azure Functions and Computer Vision.

## Clean up resources

If you're not going to continue to use this application, you can delete the resources you created by removing the resource group.

1. Select **Resource groups** from the main navigation
2. Select the `msdocs-storage-function` resource group from the list.
3. Select the **Delete resource group** button at the top of the resource group overview page.
4. Enter the resource group name `msdocs-storage-function` in the confirmation dialog.
5. Select **delete**. The process to delete the resource group may take a few minutes to complete.

# JavaScript Tutorial: Upload and analyze a file with Azure Functions and Blob Storage

8/22/2022 • 13 minutes to read • [Edit Online](#)

In this tutorial, you'll learn how to upload an image to Azure Blob Storage and process it using Azure Functions and Computer Vision. You'll also learn how to implement Azure Function triggers and bindings as part of this process. Together, these services will analyze an uploaded image that contains text, extract the text out of it, and then store the text in a database row for later analysis or other purposes.

Azure Blob Storage is Microsoft's massively scalable object storage solution for the cloud. Blob Storage is designed for storing images and documents, streaming media files, managing backup and archive data, and much more. You can read more about Blob Storage on the [overview page](#).

Azure Functions is a serverless computer solution that allows you to write and run small blocks of code as highly scalable, serverless, event driven functions. You can read more about Azure Functions on the [overview page](#).

In this tutorial, you'll learn how to:

- Upload images and files to Blob Storage
- Use an Azure Function event trigger to process data uploaded to Blob Storage
- Use Cognitive Services to analyze an image
- Write data to Table Storage using Azure Function output bindings

## Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- [Visual Studio Code](#) installed.
  - [Azure Functions extension](#) to deploy and configure the Function App.
  - [Azure Storage extension](#)
  - [Azure Resources extension](#)

## Create the storage account and container

The first step is to create the storage account that will hold the uploaded blob data, which in this scenario will be images that contain text. A storage account offers several different services, but this tutorial utilizes Blob Storage and Table Storage.

- [Visual Studio Code](#)
  - [Azure portal](#)
  - [Azure CLI](#)
1. In Visual Studio Code, select **Ctrl + Shift + P** to open the command palette.
  2. Search for **Azure Storage: Create Storage Account (Advanced)**.
  3. Use the following table to create the Storage resource.

SETTING	VALUE
Name	Enter <code>msdocsstoragefunction</code> or something similar.
Resource Group	Create the <code>msdocs-storage-function</code> resource group you created earlier.
Static web hosting	No.
Location	Choose the region closest to you.

4. In Visual Studio Code, select Shift + Alt + A to open the **Azure Explorer**.
5. Expand the **Storage** section, expand your subscription node and wait for the resource to be created.

### Create the container in Visual Studio Code

1. Still in the Azure Explorer with your new Storage resource found, expand the resource to see the nodes.
2. Right-click on **Blob Containers** and select **Create Blob Container**.
3. Enter the name `imageanalysis`. This creates a private container.

### Change from private to public container in Azure portal

This procedure expects a public container. To change that configuration, make the change in the Azure portal.

1. Right-click on the Storage Resource in the Azure Explorer and select **Open in Portal**.
2. In the **Data Storage** section, select **Containers**.
3. Find your container, `imageanalysis`, and select the `...` (ellipse) at the end of the line.
4. Select **Change access level**.
5. Select **Blob (anonymous read access for blobs only)** then select **Ok**.
6. Return to Visual Studio Code.

### Retrieve the connection string in Visual Studio Code

1. In Visual Studio Code, select Shift + Alt + A to open the **Azure Explorer**.
2. Right-click on your storage resource and select **Copy Connection String**.
3. paste this somewhere to use for later.
4. Also make note of the storage account name `msdocsstoragefunction` for later as well.

## Create the Computer Vision service

Next, create the Computer Vision service account that will process our uploaded files. Computer Vision is part of Azure Cognitive Services and offers various features for extracting data out of images. You can learn more about Computer Vision on the [overview page](#).

- [Azure portal](#)
- [Azure CLI](#)

1. In the search bar at the top of the portal, search for *Computer* and select the result labeled **Computer vision**.
2. On the **Computer vision** page, select **+ Create**.
3. On the **Create Computer Vision** page, enter the following values:
  - **Subscription:** Choose your desired Subscription.
  - **Resource Group:** Use the `msdocs-storage-function` resource group you created earlier.

- **Region:** Select the region that is closest to you.
- **Name:** Enter in a name of `msdocscomputervision`.
- **Pricing Tier:** Choose **Free** if it's available, otherwise choose **Standard S1**.
- Check the **Responsible AI Notice** box if you agree to the terms

Home > Cognitive Services >  
Create Computer Vision ...

**Basics** Network Identity Tags Review + create

Boost content discoverability, accelerate text extraction, and create products that more people can use by embedding vision capabilities in your apps. Use visual data processing to label content (from objects to concepts), extract printed and handwritten text, recognize familiar subjects like brands and landmarks, and moderate content. No machine learning expertise is required.

[Learn more](#)

**Project Details**

Subscription \* `C&L Cross Service Content Team Testing`  
Resource group \* `msdocs-storage-function`  
[Create new](#)

**Instance Details**

Region \* `East US`  
Name \* `msdocscomputervision`

Pricing tier \* `Free F0 (20 Calls per minute, 5K Calls per month)`

[View full pricing details](#)

**Responsible AI Notice**

Microsoft provides technical documentation regarding the appropriate operation applicable to this Cognitive Service that is made available by Microsoft. Customer acknowledges and agrees that they have reviewed this documentation and will use this service in accordance with it. This Cognitive Services is intended to process Customer Data that includes Biometric Data (as may be further described in product documentation) that Customer may incorporate into its own systems used for personal identification or other purposes. Customer acknowledges and agrees that it is responsible for complying with the Biometric Data obligations contained in the Online Services DPA.

[Online Services DPA](#)

Responsible Use of AI documentation for Spatial Analysis

By checking this box I certify that I have  reviewed and acknowledge all the terms above.

4. Select **Review + Create** at the bottom. Azure will take a moment validate the information you entered. Once the settings are validated, choose **Create** and Azure will begin provisioning the Computer Vision service, which might take a moment.

5. When the operation has completed, select **Go to Resource**.

## Retrieve the keys

Next, we need to find the secret key and endpoint URL for the Computer Vision service to use in our Azure Function app.

1. On the **Computer Vision** overview page, select **Keys and Endpoint**.
2. On the **Keys and EndPoint** page, copy the **Key 1** value and the **EndPoint** values and paste them somewhere to use for later. The endpoint should be in the format of

`https://YOUR-RESOURCE-NAME.cognitiveservices.azure.com/`

The screenshot shows the Azure portal interface for managing a Cognitive Services Computer Vision API. The left sidebar lists various management options like Overview, Activity log, Access control (IAM), Tags, and Diagnose and solve problems. Under Resource Management, the 'Keys and Endpoint' option is selected and highlighted with a red box. The main content area shows a tooltip about the importance of securely storing keys. Below the tooltip, there are four input fields: 'KEY 1' (redacted), 'KEY 2' (redacted), 'Location/Region' set to 'eastus', and 'Endpoint' set to 'https://msdocscomputervision.cognitiveservices.azure.com/'. Each of these last three fields is also highlighted with a red box.

## Download and configure the sample project

The code for the Azure Function used in this tutorial can be found in [this GitHub repository](#), in the JavaScript subdirectory. You can also clone the project using the command below.

```
git clone https://github.com/Azure-Samples/msdocs-storage-bind-function-service.git \
cd msdocs-storage-bind-function-service/javascript \
code .
```

The sample project accomplishes the following tasks:

- Retrieves environment variables to connect to the storage account and Computer Vision service
- Accepts the uploaded file as a blob parameter
- Analyzes the blob using the Computer Vision service
- Sends the analyzed image text to a new table row using output bindings

Once you've downloaded and opened the project, there are a few essential concepts to understand:

CONCEPT	PURPOSE
Function	The Azure Function is defined by both the function code and the bindings. The function code is in <a href="#">./ProcessImageUpload/index.js</a> .
Triggers and bindings	The triggers and bindings indicate that data which is expected into or out of the function and which service is going to send or receive that data. The trigger and bindings for this function is in <a href="#">./ProcessImageUpload/function.json</a> .

### Triggers and bindings

The following [function.json](#) file defines the triggers and bindings for this function:

```
{
 "bindings": [
 {
 "name": "myBlob",
 "type": "blobTrigger",
 "direction": "in",
 "path": "imageanalysis/{name}",
 "connection": "StorageConnection"
 },
 {
 "tableName": "ImageText",
 "connection": "StorageConnection",
 "name": "tableBinding",
 "type": "table",
 "direction": "out"
 }
]
}
```

- **Data In** - The **BlobTrigger** (`"type": "blobTrigger"`) is used to bind the function to the upload event in Blob Storage. The trigger has two required parameters:

- `path` : The path the trigger watches for events. The path includes the container name, `imageanalysis`, and the variable substitution for the blob name. This blob name is retrieved from the `name` property.
- `name` : The name of the blob uploaded. The use of the `myBlob` is the parameter name for the blob coming into the function. Don't change the value `myBlob`.
- `connection` : The **connection string** of the storage account. The value `StorageConnection` matches the name in the `local.settings.json` file.

- **Data Out** - The **TableBinding** (`"type": "table"`) is used to bind the outbound data to a Storage table.

- `tableName` : The name of the table to write the parsed image text value returned by the function. The table must already exist.
- `connection` : The **connection string** of the storage account. The value `StorageConnection` matches the name in the `local.settings.json` file.

```
// ProcessImageUpload/index.js
const { v4: uuidv4 } = require('uuid');
const { ApiKeyCredentials } = require('@azure/ms-rest-js');
const { ComputerVisionClient } = require('@azure/cognitiveservices-computervision');
const sleep = require('util').promisify(setTimeout);

const STATUS_SUCCEEDED = "succeeded";
const STATUS_FAILED = "failed"

async function readImageUrl(context, computerVisionClient, url) {

 try {

 context.log(`uri = ${url}`);

 // To recognize text in a local image, replace client.read() with readTextInStream() as shown:
 let result = await computerVisionClient.read(url);

 // Operation ID is last path segment of operationLocation (a URL)
 let operation = result.operationLocation.split('/').slice(-1)[0];

 // Wait for read recognition to complete
 // result.status is initially undefined, since it's the result of read
 while (result.status !== STATUS_SUCCEEDED) {
 await sleep(1000);
 }

 return result.readResult[0].textAnnotations;
 }
}
```

```

 result = await computervisionClient.getReadResult(operation);
 }

 let contents = "";

 result.analyzeResult.readResults.map((page) => {
 page.lines.map(line => {
 contents += line.text + "\n\r"
 });
 });
 return contents;
}

} catch (err) {
 console.log(err);
}
}

module.exports = async function (context, myBlob) {

 try {
 context.log("JavaScript blob trigger function processed blob \n Blob:",
 context.bindingData.blobTrigger, "\n Blob Size:", myBlob.length, "Bytes");

 const computerVision_ResourceKey = process.env.ComputerVisionKey;
 const computerVision_Endpoint = process.env.ComputerVisionEndPoint;

 const computerVisionClient = new ComputerVisionClient(
 new ApiKeyCredentials({ inHeader: { 'Ocp-Apim-Subscription-Key': computerVision_ResourceKey } })
), computerVision_Endpoint);

 // URL must be full path
 const textContext = await readImageUrl(context, computerVisionClient, context.bindingData.uri);

 context.bindings.tableBinding = [];
 context.bindings.tableBinding.push({
 PartitionKey: "Images",
 RowKey: uuidv4().toString(),
 Text: textContext
 });
 } catch (err) {
 context.log(err);
 return;
 }
};

};

```

This code also retrieves essential configuration values from environment variables, such as the Blob Storage connection string and Computer Vision key. These environment variables are added to the Azure Function environment after it's deployed.

The default function also utilizes a second method called `AnalyzeImage`. This code uses the URL Endpoint and Key of the Computer Vision account to make a request to Computer Vision to process the image. The request returns all of the text discovered in the image. This text is written to Table Storage, using the outbound binding.

### Configure local settings

To run the project locally, enter the environment variables in the `./local.settings.json` file. Fill in the placeholder values with the values you saved earlier when creating the Azure resources.

Although the Azure Function code runs locally, it connects to the cloud-based services for Storage, rather than using any local emulators.

```
{
 "IsEncrypted": false,
 "Values": {
 "AzureWebJobsStorage": "",
 "FUNCTIONS_WORKER_RUNTIME": "node",
 "StorageConnection": "your-storage-account-connection-string",
 "StorageAccountName": "your-storage-account-name",
 "StorageContainerName": "your-storage-container-name",
 "ComputerVisionKey": "your-computer-vision-key",
 "ComputerVisionEndPoint": "https://REPLACE-WITH-YOUR-RESOURCE-NAME.cognitiveservices.azure.com/"
 }
}
```

## Create Azure Functions app

You're now ready to deploy the application to Azure using a Visual Studio Code extension.

1. In Visual Studio Code, select Shift + Alt + A to open the **Azure** explorer.
2. In the **Functions** section, find and right-click the subscription, and select **Create Function App in Azure (Advanced)**.
3. Use the following table to create the Function resource.

SETTING	VALUE
Name	Enter <i>msdocsprocessimage</i> or something similar.
Runtime stack	Select a <b>Node.js LTS</b> version.
OS	Select <b>Linux</b> .
Resource Group	Choose the <b>msdocs-storage-function</b> resource group you created earlier.
Location	Choose the region closest to you.
Plan Type	Select <b>Consumption</b> .
Azure Storage	Select the storage account you created earlier.
Application Insights	Skip for now.

4. Azure provisions the requested resources, which will take a few moments to complete.

## Deploy Azure Functions app

1. When the previous resource creation process finishes, right-click the new resource in the **Functions** section of the Azure explorer, and select **Deploy to Function App**.
2. If asked **Are you sure you want to deploy...**, select **Deploy**.
3. When the process completes, a notification appears which a choice which includes **Upload settings**. Select that option. This copies the values from your local.settings.json file into your Azure Function app. If the notification disappeared before you could select it, continue to the next section.

## Add app settings for Storage and Computer Vision

If you selected **Upload settings** in the notification, skip this section.

The Azure Function was deployed successfully, but it can't connect to our Storage account and Computer Vision services yet. The correct keys and connection strings must first be added to the configuration settings of the Azure Functions app.

1. Find your resource in the **Functions** section of the Azure explorer, right-click **Application Settings**, and select **Add New Setting**.
2. Enter a new app setting for the following secrets. Copy and paste your secret values from your local project in the `local.settings.json` file.

SETTING
StorageConnection
StorageAccountName
StorageContainerName
ComputerVisionKey
ComputerVisionEndPoint

All of the required environment variables to connect our Azure function to different services are now in place.

## Upload an image to Blob Storage

You're now ready to test out our application! You can upload a blob to the container, and then verify that the text in the image was saved to Table Storage.

1. In the Azure explorer in Visual Studio Code, find and expand your Storage resource in the **Storage** section.
2. Expand **Blob Containers** and right-click your container name, `imageanalysis`, then select **Upload files**.
3. You can find a few sample images included in the **images** folder at the root of the downloadable sample project, or you can use one of your own.
4. For the **Destination directory**, accept the default value, `/`.
5. Wait until the files are uploaded and listed in the container.

## View text analysis of image

Next, you can verify that the upload triggered the Azure Function, and that the text in the image was analyzed and saved to Table Storage properly.

1. In Visual Studio Code, in the Azure Explorer, under the same Storage resource, expand **Tables** to find your resource.
2. An **ImageText** table should now be available. Click on the table to preview the data rows inside of it. You should see an entry for the processed image text of an uploaded file. You can verify this using either the **Timestamp**, or by viewing the content of the **Text** column.

Congratulations! You succeeded in processing an image that was uploaded to Blob Storage using Azure Functions and Computer Vision.

## Troubleshooting

Please use the following table to help troubleshoot issues during this procedure.

ISSUE	RESOLUTION
<pre>await computerVisionClient.read(url); errors with Only absolute URLs are supported</pre>	Make sure your <code>ComputerVisionEndPoint</code> endpoint is in the format of <code>https://YOUR-RESOURCE-NAME.cognitiveservices.azure.com/</code>

## Clean up resources

If you're not going to continue to use this application, you can delete the resources you created by removing the resource group.

1. Select **Resource groups** from the Azure explorer
2. Find and right-click the `msdocs-storage-function` resource group from the list.
3. Select **Delete**. The process to delete the resource group may take a few minutes to complete.

# Tutorial: Migrate on-premises data to cloud storage with AzCopy

8/22/2022 • 6 minutes to read • [Edit Online](#)

AzCopy is a command-line tool for copying data to or from Azure Blob storage, Azure Files, and Azure Table storage, by using simple commands. The commands are designed for optimal performance. Using AzCopy, you can either copy data between a file system and a storage account, or between storage accounts. AzCopy may be used to copy data from local (on-premises) data to a storage account.

In this tutorial, you learn how to:

- Create a storage account.
- Use AzCopy to upload all your data.
- Modify the data for test purposes.
- Create a scheduled task or cron job to identify new files to upload.

If you don't have an Azure subscription, create a [free account](#) before you begin.

## Prerequisites

To complete this tutorial, download the latest version of AzCopy. See [Get started with AzCopy](#).

If you're on Windows, you will require [Schtasks](#) as this tutorial makes use of it in order to schedule a task. Linux users will make use of the crontab command, instead.

To create a general-purpose v2 storage account in the Azure portal, follow these steps:

1. Under **Azure services**, select **Storage accounts**.
2. On the **Storage Accounts** page, choose **+ Create**.
3. On the **Basics** blade, select the subscription in which to create the storage account.
4. Under the **Resource group** field, select your desired resource group, or create a new resource group. For more information on Azure resource groups, see [Azure Resource Manager overview](#).
5. Next, enter a name for your storage account. The name you choose must be unique across Azure. The name also must be between 3 and 24 characters in length, and may include only numbers and lowercase letters.
6. Select a region for your storage account, or use the default region.
7. Select a performance tier. The default tier is *Standard*.
8. Specify how the storage account will be replicated. The default redundancy option is *Geo-redundant storage (GRS)*. For more information about available replication options, see [Azure Storage redundancy](#).
9. Additional options are available on the **Advanced**, **Networking**, **Data protection**, and **Tags** blades. To use Azure Data Lake Storage, choose the **Advanced** blade, and then set **Hierarchical namespace** to **Enabled**. For more information, see [Azure Data Lake Storage Gen2 Introduction](#).
10. Select **Review + Create** to review your storage account settings and create the account.
11. Select **Create**.

The following image shows the settings on the **Basics** blade for a new storage account:

The screenshot shows the Microsoft Azure portal interface for creating a new storage account. The left sidebar lists various services like Home, All services, and Storage accounts. The main content area is titled 'Create a storage account' and is currently on the 'Basics' tab. It provides a brief overview of Azure Storage and its features. In the 'Project details' section, the subscription is set to 'My Example Subscription' and the resource group is 'myexamplegroup'. Under 'Instance details', the storage account name is 'theexampleaccount', it's located in the '(US) East US' region, and it uses 'Standard' performance and 'Geo-redundant storage (GRS)' redundancy. A checkbox for 'Make read access to data available in the event of regional unavailability.' is checked. At the bottom, there are navigation buttons for 'Review + create', '< Previous', and 'Next : Advanced >'.

## Create a container

The first step is to create a container, because blobs must always be uploaded into a container. Containers are used as a method of organizing groups of blobs like you would files on your computer, in folders.

Follow these steps to create a container:

1. Select the **Storage accounts** button from the main page, and select the storage account that you created.
2. Select **Blobs** under **Services**, and then select **Container**.

The screenshot shows the Microsoft Azure portal interface for managing blobs in the 'rags' storage account. The left sidebar lists various services. The main content area shows the 'rags - Blobs' storage account. On the right, the 'Containers' blade is open, displaying a table with one row named 'ragrs'. A modal window titled 'New container' is overlaid, asking for a 'Name' (marked with a red asterisk) and a 'Public access level' (set to 'Private (no anonymous access)'). There are 'OK' and 'Cancel' buttons at the bottom of the modal. The left sidebar also shows the 'Storage accounts' section with 'rags' selected.

Container names must start with a letter or number. They can contain only letters, numbers, and the hyphen character (-). For more rules about naming blobs and containers, see [Naming and referencing containers, blobs, and metadata](#).

## Download AzCopy

Download the AzCopy V10 executable file.

- [Windows](#) (zip)
- [Linux](#) (tar)
- [macOS](#) (zip)

Place the AzCopy file anywhere on your computer. Add the location of the file to your system path variable so that you can refer to this executable file from any folder on your computer.

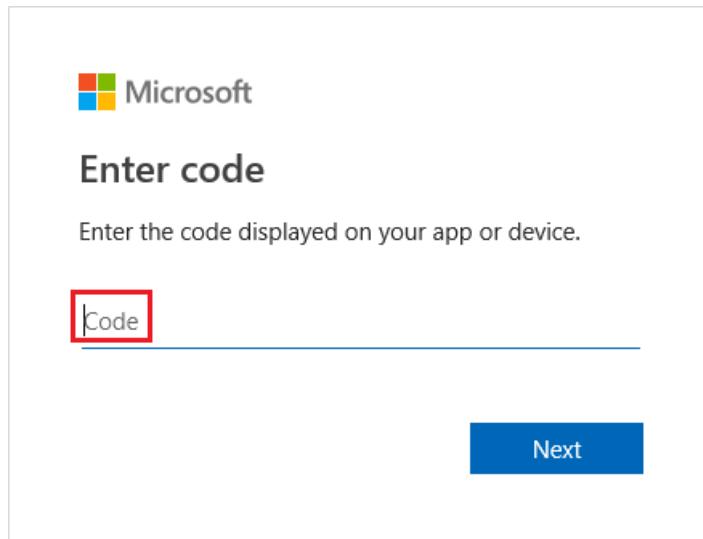
## Authenticate with Azure AD

First, assign the [Storage Blob Data Contributor](#) role to your identity. See [Assign an Azure role for access to blob data](#).

Then, open a command prompt, type the following command, and press the ENTER key.

```
azcopy login
```

This command returns an authentication code and the URL of a website. Open the website, provide the code, and then choose the **Next** button.



A sign-in window will appear. In that window, sign into your Azure account by using your Azure account credentials. After you've successfully signed in, you can close the browser window and begin using AzCopy.

## Upload contents of a folder to Blob storage

You can use AzCopy to upload all files in a folder to Blob storage on [Windows](#) or [Linux](#). To upload all blobs in a folder, enter the following AzCopy command:

```
azcopy copy "<local-folder-path>" "https://<storage-account-name>.<blob or dfs>.core.windows.net/<container-name>" --recursive=true
```

- Replace the `<local-folder-path>` placeholder with the path to a folder that contains files (For example:

`C:\myFolder` or `/mnt/myFolder` ).

- Replace the `<storage-account-name>` placeholder with the name of your storage account.
- Replace the `<container-name>` placeholder with the name of the container that you created.

To upload the contents of the specified directory to Blob storage recursively, specify the `--recursive` option. When you run AzCopy with this option, all subfolders and their files are uploaded as well.

## Upload modified files to Blob storage

You can use AzCopy to upload files based on their last-modified time.

To try this, modify or create new files in your source directory for test purposes. Then, use the AzCopy `sync` command.

```
azcopy sync "<local-folder-path>" "https://<storage-account-name>.blob.core.windows.net/<container-name>" --recursive=true
```

- Replace the `<local-folder-path>` placeholder with the path to a folder that contains files (For example: `C:\myFolder` or `/mnt/myFolder` ).
- Replace the `<storage-account-name>` placeholder with the name of your storage account.
- Replace the `<container-name>` placeholder with the name of the container that you created.

To learn more about the `sync` command, see [Synchronize files](#).

## Create a scheduled task

You can create a scheduled task or cron job that runs an AzCopy command script. The script identifies and uploads new on-premises data to cloud storage at a specific time interval.

Copy the AzCopy command to a text editor. Update the parameter values of the AzCopy command to the appropriate values. Save the file as `script.sh` (Linux) or `script.bat` (Windows) for AzCopy.

These examples assume that your folder is named `myFolder`, your storage account name is `mystorageaccount` and your container name is `mycontainer`.

### NOTE

The Linux example appends a SAS token. You'll need to provide one in your command. The current version of AzCopy V10 doesn't support Azure AD authorization in cron jobs.

- [Linux](#)
- [Windows](#)

```
azcopy sync "/mnt/myfiles" "https://mystorageaccount.blob.core.windows.net/mycontainer?sv=2018-03-28&ss=bfqt&srt=sco&sp=rwdlacup&se=2019-05-30T06:57:40Z&st=2019-05-29T22:57:40Z&spr=https&sig=BXHippZxxx54hQn%2F4tBY%2BE2JHGCTRv52445rtoyqgFBUo%3D" --recursive=true
```

In this tutorial, [Schtasks](#) is used to create a scheduled task on Windows. The [Crontab](#) command is used to create a cron job on Linux.

[Schtasks](#) enables an administrator to create, delete, query, change, run, and end scheduled tasks on a local or

remote computer. **Cron** enables Linux and Unix users to run commands or scripts at a specified date and time by using [cron expressions](#).

- [Linux](#)
- [Windows](#)

To create a cron job on Linux, enter the following command on a terminal:

```
crontab -e
*/5 * * * * sh /path/to/script.sh
```

Specifying the cron expression `*/5 * * * *` in the command indicates that the shell script `script.sh` should run every five minutes. You can schedule the script to run at a specific time daily, monthly, or yearly. To learn more about setting the date and time for job execution, see [cron expressions](#).

To validate that the scheduled task/cron job runs correctly, create new files in your `myFolder` directory. Wait five minutes to confirm that the new files have been uploaded to your storage account. Go to your log directory to view output logs of the scheduled task or cron job.

## Next steps

To learn more about ways to move on-premises data to Azure Storage and vice versa, follow this link:

- [Move data to and from Azure Storage](#).

For more information about AzCopy, see any of these articles:

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and blob storage](#)
- [Transfer data with AzCopy and file storage](#)
- [Transfer data with AzCopy and Amazon S3 buckets](#)
- [AzCopy configuration settings](#)
- [Optimize the performance of AzCopy](#)
- [Find errors and resume jobs by using log and plan files in AzCopy](#)
- [Troubleshoot problems with AzCopy v10](#)

# Create a virtual machine and storage account for a scalable application

8/22/2022 • 4 minutes to read • [Edit Online](#)

This tutorial is part one of a series. This tutorial shows you deploy an application that uploads and download large amounts of random data with an Azure storage account. When you're finished, you have a console application running on a virtual machine that you upload and download large amounts of data to a storage account.

In part one of the series, you learn how to:

- Create a storage account
- Create a virtual machine
- Configure a custom script extension

If you don't have an Azure subscription, create a [free account](#) before you begin.

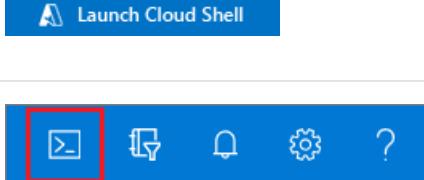
## NOTE

This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

## Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article, without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select Try It in the upper-right corner of a code block. Selecting Try It doesn't automatically copy the code to Cloud Shell.	
Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	
Select the Cloud Shell button on the menu bar at the upper right in the Azure portal.	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux, or by

selecting **Cmd+Shift+V** on macOS.

4. Select **Enter** to run the code.

If you choose to install and use the PowerShell locally, this tutorial requires the Azure PowerShell module Az version 0.7 or later. Run `Get-Module -ListAvailable Az` to find the version. If you need to upgrade, see [Install Azure PowerShell module](#). If you are running PowerShell locally, you also need to run `Connect-AzAccount` to create a connection with Azure.

## Create a resource group

Create an Azure resource group with [New-AzResourceGroup](#). A resource group is a logical container into which Azure resources are deployed and managed.

```
New-AzResourceGroup -Name myResourceGroup -Location EastUS
```

## Create a storage account

The sample uploads 50 large files to a blob container in an Azure Storage account. A storage account provides a unique namespace to store and access your Azure storage data objects. Create a storage account in the resource group you created by using the [New-AzStorageAccount](#) command.

In the following command, substitute your own globally unique name for the Blob storage account where you see the `<blob_storage_account>` placeholder.

```
$storageAccount = New-AzStorageAccount -ResourceGroupName myResourceGroup `
-Name "<blob_storage_account>" `
-Location EastUS `
-SkuName Standard_LRS `
-Kind Storage `
```

## Create a virtual machine

Create a virtual machine configuration. This configuration includes the settings that are used when deploying the virtual machine such as a virtual machine image, size, and authentication configuration. When running this step, you are prompted for credentials. The values that you enter are configured as the user name and password for the virtual machine.

Create the virtual machine with [New-AzVM](#).

```

Variables for common values
$resourceGroup = "myResourceGroup"
$location = "eastus"
$vmName = "myVM"

Create user object
$cred = Get-Credential -Message "Enter a username and password for the virtual machine."

Create a subnet configuration
$subnetConfig = New-AzVirtualNetworkSubnetConfig -Name mySubnet -AddressPrefix 192.168.1.0/24

Create a virtual network
$vnet = New-AzVirtualNetwork -ResourceGroupName $resourceGroup -Location $location `
 -Name MYvNET -AddressPrefix 192.168.0.0/16 -Subnet $subnetConfig

Create a public IP address and specify a DNS name
$pip = New-AzPublicIpAddress -ResourceGroupName $resourceGroup -Location $location `
 -Name "mypublicdns$(Get-Random)" -AllocationMethod Static -IdleTimeoutInMinutes 4

Create a virtual network card and associate with public IP address
$nic = New-AzNetworkInterface -Name myNic -ResourceGroupName $resourceGroup -Location $location `
 -SubnetId $vnet.Subnets[0].Id -PublicIpAddressId $pip.Id

Create a virtual machine configuration
$vmConfig = New-AzVMConfig -VMName myVM -VMSize Standard_DS14_v2 | `
 Set-AzVMOperatingSystem -Windows -ComputerName myVM -Credential $cred | `
 Set-AzVMSourceImage -PublisherName MicrosoftWindowsServer -Offer WindowsServer `
 -Skus 2016-Datacenter -Version latest | Add-AzVMNetworkInterface -Id $nic.Id

Create a virtual machine
New-AzVM -ResourceGroupName $resourceGroup -Location $location -VM $vmConfig

Write-host "Your public IP address is $($pip.IpAddress)"

```

## Deploy configuration

For this tutorial, there are pre-requisites that must be installed on the virtual machine. The custom script extension is used to run a PowerShell script that completes the following tasks:

- Install .NET core 2.0
- Install chocolatey
- Install GIT
- Clone the sample repo
- Restore NuGet packages
- Creates 50 1-GB files with random data

Run the following cmdlet to finalize configuration of the virtual machine. This step takes 5-15 minutes to complete.

```

Start a CustomScript extension to use a simple PowerShell script to install .NET core, dependencies, and
pre-create the files to upload.
Set-AzVMCustomScriptExtension -ResourceGroupName myResourceGroup `
 -VMName myVM `
 -Location EastUS `
 -FileUri https://raw.githubusercontent.com/azure-samples/storage-dotnet-perf-scale-`
 app/master/setup_env.ps1 `
 -Run 'setup_env.ps1' `
 -Name DemoScriptExtension

```

## Next steps

In part one of the series, you learned about creating a storage account, deploying a virtual machine and configuring the virtual machine with the required pre-requisites such as how to:

- Create a storage account
- Create a virtual machine
- Configure a custom script extension

Advance to part two of the series to upload large amounts of data to a storage account using exponential retry and parallelism.

[Upload large amounts of large files in parallel to a storage account](#)

# Upload large amounts of random data in parallel to Azure storage

8/22/2022 • 7 minutes to read • [Edit Online](#)

This tutorial is part two of a series. This tutorial shows you deploy an application that uploads large amount of random data to an Azure storage account.

In part two of the series, you learn how to:

- Configure the connection string
- Build the application
- Run the application
- Validate the number of connections

Microsoft Azure Blob Storage provides a scalable service for storing your data. To ensure your application is as performant as possible, an understanding of how blob storage works is recommended. Knowledge of the limits for Azure blobs is important, to learn more about these limits visit: [Scalability and performance targets for Blob storage](#).

[Partition naming](#) is another potentially important factor when designing a high-performance application using blobs. For block sizes greater than or equal to 4 MiB, [High-Throughput block blobs](#) are used, and partition naming will not impact performance. For block sizes less than 4 MiB, Azure storage uses a range-based partitioning scheme to scale and load balance. This configuration means that files with similar naming conventions or prefixes go to the same partition. This logic includes the name of the container that the files are being uploaded to. In this tutorial, you use files that have GUIDs for names as well as randomly generated content. They are then uploaded to five different containers with random names.

## Prerequisites

To complete this tutorial, you must have completed the previous Storage tutorial: [Create a virtual machine and storage account for a scalable application](#).

## Remote into your virtual machine

Use the following command on your local machine to create a remote desktop session with the virtual machine. Replace the IP address with the publicIPAddress of your virtual machine. When prompted, enter the credentials you used when creating the virtual machine.

```
mstsc /v:<publicIpAddress>
```

## Configure the connection string

In the Azure portal, navigate to your storage account. Select **Access keys** under **Settings** in your storage account. Copy the **connection string** from the primary or secondary key. Log in to the virtual machine you created in the previous tutorial. Open a **Command Prompt** as an administrator and run the `setx` command with the `/m` switch, this command saves a machine setting environment variable. The environment variable is not available until you reload the **Command Prompt**. Replace `<storageConnectionString>` in the following sample:

```
setx storageconnectionstring "<storageConnectionString>" /m
```

When finished, open another **Command Prompt**, navigate to `D:\git\storage-dotnet-perf-scale-app` and type `dotnet build` to build the application.

## Run the application

Navigate to `D:\git\storage-dotnet-perf-scale-app`.

Type `dotnet run` to run the application. The first time you run `dotnet` it populates your local package cache, to improve restore speed and enable offline access. This command takes up to a minute to complete and only happens once.

```
dotnet run
```

The application creates five randomly named containers and begins uploading the files in the staging directory to the storage account.

The `uploadFilesAsync` method is shown in the following example:

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

```
private static async Task UploadFilesAsync()
{
 // Create five randomly named containers to store the uploaded files.
 BlobContainerClient[] containers = await GetRandomContainersAsync();

 // Path to the directory to upload
 string uploadPath = Directory.GetCurrentDirectory() + "\\upload";

 // Start a timer to measure how long it takes to upload all the files.
 Stopwatch timer = Stopwatch.StartNew();

 try
 {
 Console.WriteLine($"Iterating in directory: {uploadPath}");
 int count = 0;

 Console.WriteLine($"Found {Directory.GetFiles(uploadPath).Length} file(s)");

 // Specify the StorageTransferOptions
 BlobUploadOptions options = new BlobUploadOptions
 {
 TransferOptions = new StorageTransferOptions
 {
 // Set the maximum number of workers that
 // may be used in a parallel transfer.
 MaximumConcurrency = 8,

 // Set the maximum length of a transfer to 50MB.
 MaximumTransferSize = 50 * 1024 * 1024
 }
 };

 // Create a queue of tasks that will each upload one file.
 var tasks = new Queue<Task<Response<BlobContentInfo>>>();

 // Iterate through the files
 foreach (string filePath in Directory.GetFiles(uploadPath))
```

```

 {
 BlobContainerClient container = containers[count % 5];
 string fileName = Path.GetFileName(filePath);
 Console.WriteLine($"Uploading {fileName} to container {container.Name}");
 BlobClient blob = container.GetBlobClient(fileName);

 // Add the upload task to the queue
 tasks.Enqueue(blob.UploadAsync(filePath, options));
 count++;
 }

 // Run all the tasks asynchronously.
 await Task.WhenAll(tasks);

 timer.Stop();
 Console.WriteLine($"Uploaded {count} files in {timer.Elapsed.TotalSeconds} seconds");
}
catch (RequestFailedException ex)
{
 Console.WriteLine($"Azure request failed: {ex.Message}");
}
catch (DirectoryNotFoundException ex)
{
 Console.WriteLine($"Error parsing files in the directory: {ex.Message}");
}
catch (Exception ex)
{
 Console.WriteLine($"Exception: {ex.Message}");
}
}
}

```

The following example is a truncated application output running on a Windows system.

```

Created container 2dbb45f4-099e-49eb-880c-5b02ebac135e
Created container 0d784365-3bdf-4ef2-b2b2-c17b6480792b
Created container 42ac67f2-a316-49c9-8fdb-860fb32845d7
Created container f0357772-cb04-45c3-b6ad-ff9b7a5ee467
Created container 92480da9-f695-4a42-abe8-fb35e71eb887
Iterating in directory: C:\git\myapp\upload
Found 5 file(s)
Uploading 1d596d16-f6de-4c4c-8058-50ebd8141e4d.pdf to container 2dbb45f4-099e-49eb-880c-5b02ebac135e
Uploading 242ff392-78be-41fb-b9d4-aee8152a6279.pdf to container 0d784365-3bdf-4ef2-b2b2-c17b6480792b
Uploading 38d4d7e2-acb4-4efc-ba39-f9611d0d55ef.pdf to container 42ac67f2-a316-49c9-8fdb-860fb32845d7
Uploading 45930d63-b0d0-425f-a766-cda27ff00d32.pdf to container f0357772-cb04-45c3-b6ad-ff9b7a5ee467
Uploading 5129b385-5781-43be-8bac-e2fbb7d2bd82.pdf to container 92480da9-f695-4a42-abe8-fb35e71eb887
Uploaded 5 files in 16.9552163 seconds

```

## Validate the connections

While the files are being uploaded, you can verify the number of concurrent connections to your storage account. Open a console window and type `netstat -a | find /c "blob:https"`. This command shows the number of connections that are currently opened. As you can see from the following example, 800 connections were open when uploading the random files to the storage account. This value changes throughout running the upload. By uploading in parallel block chunks, the amount of time required to transfer the contents is greatly reduced.

```

C:>netstat -a | find /c "blob:https"
800

C:>

```

## Next steps

In part two of the series, you learned about uploading large amounts of random data to a storage account in parallel, such as how to:

- Configure the connection string
- Build the application
- Run the application
- Validate the number of connections

Advance to part three of the series to download large amounts of data from a storage account.

[Download large amounts of random data from Azure storage](#)

# Download large amounts of random data from Azure storage

8/22/2022 • 6 minutes to read • [Edit Online](#)

This tutorial is part three of a series. This tutorial shows you how to download large amounts of data from Azure storage.

In part three of the series, you learn how to:

- Update the application
- Run the application
- Validate the number of connections

## Prerequisites

To complete this tutorial, you must have completed the previous Storage tutorial: [Upload large amounts of random data in parallel to Azure storage](#).

## Remote into your virtual machine

To create a remote desktop session with the virtual machine, use the following command on your local machine. Replace the IP address with the publicIPAddress of your virtual machine. When prompted, enter the credentials used when creating the virtual machine.

```
mstsc /v:<publicIpAddress>
```

## Update the application

In the previous tutorial, you only uploaded files to the storage account. Open

`D:\git\storage-dotnet-perf-scale-app\Program.cs` in a text editor. Replace the `Main` method with the following sample. This example comments out the upload task and uncomments the download task and the task to delete the content in the storage account when complete.

```

public static void Main(string[] args)
{
 Console.WriteLine("Azure Blob storage performance and scalability sample");
 // Set threading and default connection limit to 100 to
 // ensure multiple threads and connections can be opened.
 // This is in addition to parallelism with the storage
 // client library that is defined in the functions below.
 ThreadPool.SetMinThreads(100, 4);
 ServicePointManager.DefaultConnectionLimit = 100; // (Or More)

 bool exception = false;
 try
 {
 // Call the UploadFilesAsync function.
 // await UploadFilesAsync();

 // Uncomment the following line to enable downloading of files from the storage account.
 // This is commented out initially to support the tutorial at
 // https://docs.microsoft.com/azure/storage/blobs/storage-blob-scalable-app-download-files
 await DownloadFilesAsync();
 }
 catch (Exception ex)
 {
 Console.WriteLine(ex.Message);
 exception = true;
 }
 finally
 {
 // The following function will delete the container and all files contained in them.
 // This is commented out initially as the tutorial at
 // https://docs.microsoft.com/azure/storage/blobs/storage-blob-scalable-app-download-files
 // has you upload only for one tutorial and download for the other.
 if (!exception)
 {
 // await DeleteExistingContainersAsync();
 }
 Console.WriteLine("Press any key to exit the application");
 Console.ReadKey();
 }
}

```

After the application has been updated, you need to build the application again. Open a `Command Prompt` and navigate to `D:\git\storage-dotnet-perf-scale-app`. Rebuild the application by running `dotnet build` as seen in the following example:

```
dotnet build
```

## Run the application

Now that the application has been rebuilt it is time to run the application with the updated code. If not already open, open a `Command Prompt` and navigate to `D:\git\storage-dotnet-perf-scale-app`.

Type `dotnet run` to run the application.

```
dotnet run
```

The `DownloadFilesAsync` task is shown in the following example:

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

The application reads the containers located in the storage account specified in the `storageConnectionString`. It iterates through the blobs using the [GetBlobs](#) method and downloads them to the local machine using the [DownloadToAsync](#) method.

```
private static async Task DownloadFilesAsync()
{
 BlobServiceClient blobServiceClient = GetBlobServiceClient();

 // Path to the directory to upload
 string downloadPath = Directory.GetCurrentDirectory() + "\\download\\";
 Directory.CreateDirectory(downloadPath);
 Console.WriteLine($"Created directory {downloadPath}");

 // Specify the StorageTransferOptions
 var options = new StorageTransferOptions
 {
 // Set the maximum number of workers that
 // may be used in a parallel transfer.
 MaximumConcurrency = 8,

 // Set the maximum length of a transfer to 50MB.
 MaximumTransferSize = 50 * 1024 * 1024
 };

 List<BlobContainerClient> containers = new List<BlobContainerClient>();

 foreach (BlobContainerItem container in blobServiceClient.GetBlobContainers())
 {
 containers.Add(blobServiceClient.GetBlobContainerClient(container.Name));
 }

 // Start a timer to measure how long it takes to download all the files.
 Stopwatch timer = Stopwatch.StartNew();

 // Download the blobs
 try
 {
 int count = 0;

 // Create a queue of tasks that will each upload one file.
 var tasks = new Queue<Task<Response>>();

 foreach (BlobContainerClient container in containers)
 {
 // Iterate through the files
 foreach (BlobItem blobItem in container.GetBlobs())
 {
 string fileName = downloadPath + blobItem.Name;
 Console.WriteLine($"Downloading {blobItem.Name} to {downloadPath}");

 BlobClient blob = container.GetBlobClient(blobItem.Name);

 // Add the download task to the queue
 tasks.Enqueue(blob.DownloadToAsync(fileName, default, options));
 count++;
 }
 }

 // Run all the tasks asynchronously.
 await Task.WhenAll(tasks);

 // Report the elapsed time.
 timer.Stop();
 Console.WriteLine($"Downloaded {count} files in {timer.Elapsed.TotalSeconds} seconds");
 }
 catch (RequestFailedException ex)
 {
```

```
 Console.WriteLine($"Azure request failed: {ex.Message}");
 }
 catch (DirectoryNotFoundException ex)
 {
 Console.WriteLine($"Error parsing files in the directory: {ex.Message}");
 }
 catch (Exception ex)
 {
 Console.WriteLine($"Exception: {ex.Message}");
 }
}
```

## Validate the connections

While the files are being downloaded, you can verify the number of concurrent connections to your storage account. Open a console window and type `netstat -a | find /c "blob:https"`. This command shows the number of connections that are currently opened. As you can see from the following example, over 280 connections were open when downloading files from the storage account.

```
C:\>netstat -a | find /c "blob:https"
289
C:\>
```

## Next steps

In part three of the series, you learned about downloading large amounts of data from a storage account, including how to:

- Run the application
- Validate the number of connections

Go to part four of the series to verify throughput and latency metrics in the portal.

[Verify throughput and latency metrics in the portal](#)

# Verify throughput and latency metrics for a storage account

8/22/2022 • 2 minutes to read • [Edit Online](#)

This tutorial is part four and the final part of a series. In the previous tutorials, you learned how to upload and download large amounts of random data to an Azure storage account. This tutorial shows you how you can use metrics to view throughput and latency in the Azure portal.

In part four of the series, you learn how to:

- Configure charts in the Azure portal
- Verify throughput and latency metrics

[Azure storage metrics](#) uses Azure monitor to provide a unified view into the performance and availability of your storage account.

## Configure metrics

Navigate to **Metrics** under **SETTINGS** in your storage account.

Choose **Blob** from the **SUB SERVICE** drop-down.

Under **METRIC**, select one of the metrics found in the following table:

The following metrics give you an idea of the latency and throughput of the application. The metrics you configure in the portal are in 1-minute averages. If a transaction finished in the middle of a minute that minute data is halved for the average. In the application, the upload and download operations were timed and provided you output of the actual amount of time it took to upload and download the files. This information can be used in conjunction with the portal metrics to fully understand throughput.

METRIC	DEFINITION
Success E2E Latency	The average end-to-end latency of successful requests made to a storage service or the specified API operation. This value includes the required processing time within Azure Storage to read the request, send the response, and receive acknowledgment of the response.
Success Server Latency	The average time used to process a successful request by Azure Storage. This value does not include the network latency specified in SuccessE2ELatency.
Transactions	The number of requests made to a storage service or the specified API operation. This number includes successful and failed requests, as well as requests that produced errors. In the example, the block size was set to 100 MB. In this case, each 100-MB block is considered a transaction.
Ingress	The amount of ingress data. This number includes ingress from an external client into Azure Storage as well as ingress within Azure.

METRIC	DEFINITION
Egress	The amount of egress data. This number includes egress from an external client into Azure Storage as well as egress within Azure. As a result, this number does not reflect billable egress.

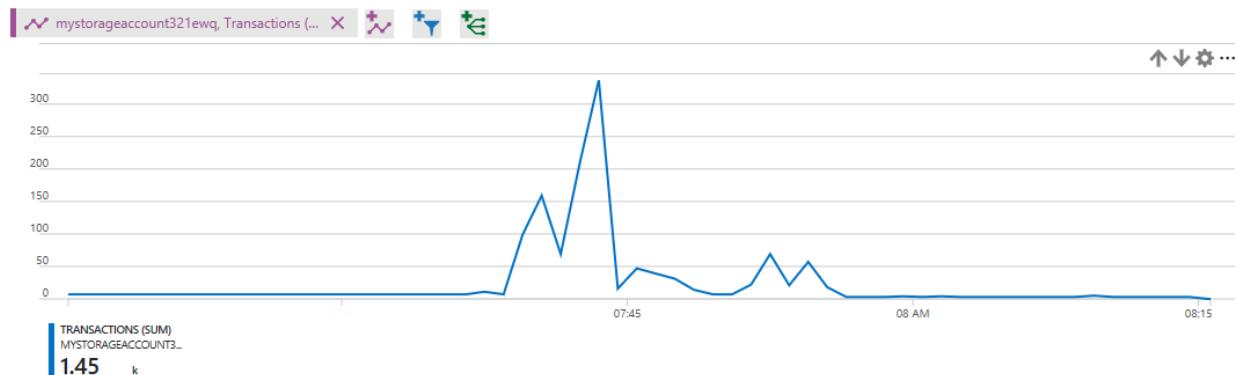
Select **Last 24 hours (Automatic)** next to Time. Choose **Last hour** and **Minute** for Time granularity, then click **Apply**.

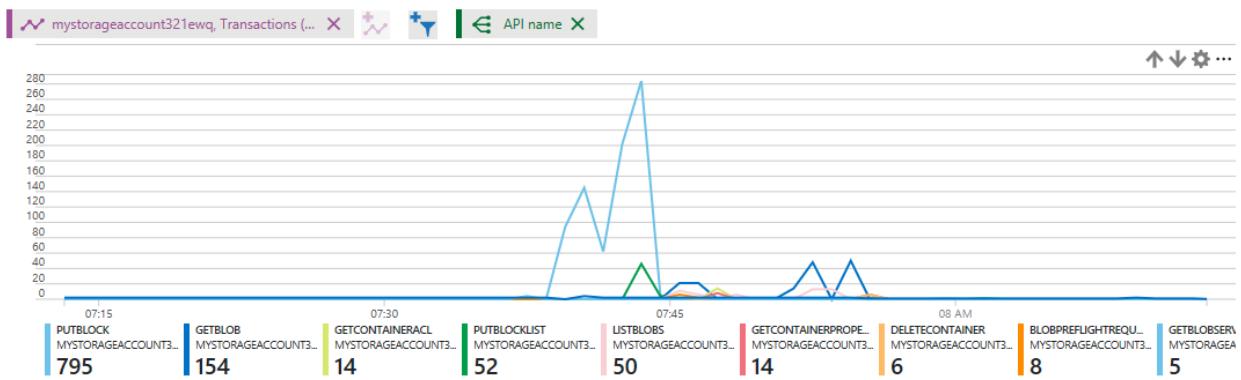
The screenshot shows the Azure Metrics (preview) interface for a storage account named 'mystorageaccount321ewq'. On the left, the navigation menu is visible with various service options like All services, Resource groups, and Storage accounts. In the center, a 'Metrics (preview)' blade is open for the storage account. A modal window titled 'Restore defaults' is displayed over the main chart area. The 'Time' section of the modal has a dropdown set to '1 Minute' under 'Time granularity'. Below it, a list of time ranges is shown, with 'Last hour' highlighted by a red box. The main chart area shows a line graph for 'Availability' over a one-hour period, with the Y-axis ranging from 0 to 100. The X-axis shows times from 07:30 to 08:15. The legend indicates the chart represents 'Availability' for the 'mystorageaccount321ewq' resource. The chart area is also outlined with a red box.

Charts can have more than one metric assigned to them, but assigning more than one metric disables the ability to group by dimensions.

## Dimensions

**Dimensions** are used to look deeper into the charts and get more detailed information. Different metrics have different dimensions. One dimension that is available is the **API name** dimension. This dimension breaks out the chart into each separate API call. The first image below shows an example chart of total transactions for a storage account. The second image shows the same chart but with the API name dimension selected. As you can see, each transaction is listed giving more details into how many calls were made by API name.





## Clean up resources

When no longer needed, delete the resource group, virtual machine, and all related resources. To do so, select the resource group for the VM and click Delete.

## Next steps

In part four of the series, you learned about viewing metrics for the example solution, such as how to:

- Configure charts in the Azure portal
- Verify throughput and latency metrics

Follow this link to see pre-built storage samples.

[Azure storage script samples](#)

# Tutorial: Host a static website on Blob Storage

8/22/2022 • 4 minutes to read • [Edit Online](#)

In this tutorial, you'll learn how to build and deploy a static website to Azure Storage. When you're finished, you will have a static website that users can access publicly.

In this tutorial, you learn how to:

- Configure static website hosting
- Deploy a Hello World website

Static websites have some limitations. For example, If you want to configure headers, you'll have to use Azure Content Delivery Network (Azure CDN). There's no way to configure headers as part of the static website feature itself. Also, AuthN and AuthZ are not supported.

If these features are important for your scenario, consider using [Azure Static Web Apps](#). It's a great alternative to static websites and is also appropriate in cases where you don't require a web server to render content. You can configure headers and AuthN / AuthZ is fully supported. Azure Static Web Apps also provides a fully managed continuous integration and continuous delivery (CI/CD) workflow from GitHub source to global deployment.

## Prerequisites

To access Azure Storage, you'll need an Azure subscription. If you don't already have a subscription, create a [free account](#) before you begin.

All access to Azure Storage takes place through a storage account. For this quickstart, create a storage account using the [Azure portal](#), Azure PowerShell, or Azure CLI. For help creating a storage account, see [Create a storage account](#).

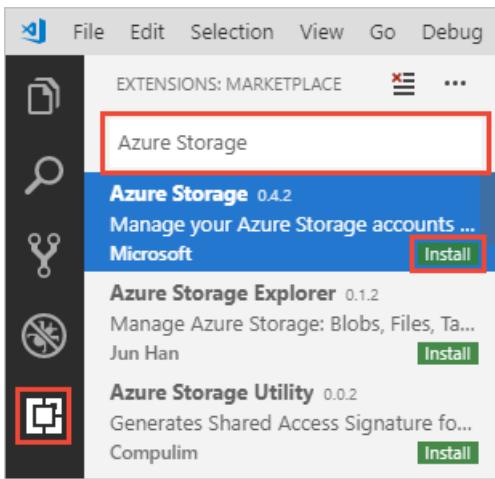
### NOTE

Static websites are now available for general-purpose v2 Standard storage accounts as well as storage accounts with hierarchical namespace enabled.

This tutorial uses [Visual Studio Code](#), a free tool for programmers, to build the static website and deploy it to an Azure Storage account.

After you install Visual Studio Code, install the Azure Storage preview extension. This extension integrates Azure Storage management functionality with Visual Studio Code. You will use the extension to deploy your static website to Azure Storage. To install the extension:

1. Launch Visual Studio Code.
2. On the toolbar, click **Extensions**. Search for *Azure Storage*, and select the **Azure Storage** extension from the list. Then click the **Install** button to install the extension.



## Sign in to the Azure portal

Sign in to the [Azure portal](#) to get started.

## Configure static website hosting

The first step is to configure your storage account to host a static website in the Azure portal. When you configure your account for static website hosting, Azure Storage automatically creates a container named `$web`. The `$web` container will contain the files for your static website.

1. Open the [Azure portal](#) in your web browser.
2. Locate your storage account and display the account overview.
3. Select **Static website** to display the configuration page for static websites.
4. Select **Enabled** to enable static website hosting for the storage account.
5. In the **Index document name** field, specify a default index page of `index.html`. The default index page is displayed when a user navigates to the root of your static website.
6. In the **Error document path** field, specify a default error page of `404.html`. The default error page is displayed when a user attempts to navigate to a page that does not exist in your static website.
7. Click **Save**. The Azure portal now displays your static website endpoint.

Microsoft Azure

Search resources, services, and docs (G+ /)

Home > contosoaccount

Contoso | Static website

Storage account

Search (Ctrl+ /) Save Discard

Data management

Geo-replication

Data protection

Object replication

Static website

Lifecycle management

Azure search

Enabling static websites on the blob service allows you to host static content. Webpages may include static content and client-side scripts. Server-side scripting is not supported. As data is replicated asynchronously from primary to secondary regions, files at the secondary endpoint may not be immediately available or in sync with files at the primary endpoint. [Learn more](#)

Static website

Disabled Enabled

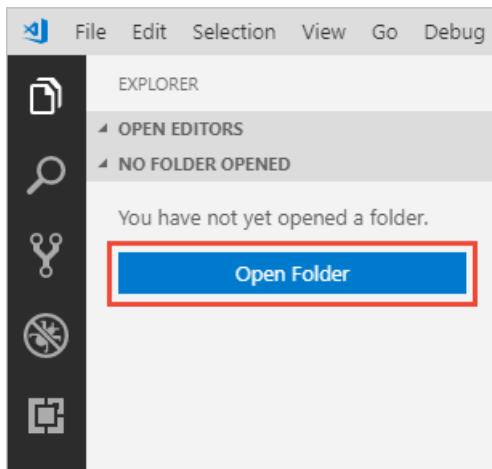
Index document name ⓘ index.html

Error document path ⓘ 404.html

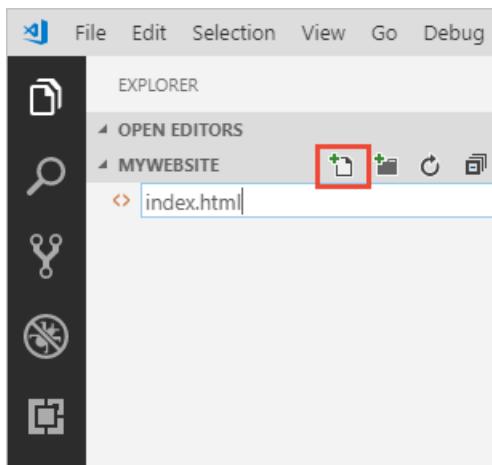
## Deploy a Hello World website

Next, create a Hello World web page with Visual Studio Code and deploy it to the static website hosted in your Azure Storage account.

1. Create an empty folder named *mywebsite* on your local file system.
2. Launch Visual Studio Code, and open the folder that you just created from the **Explorer** panel.



3. Create the default index file in the *mywebsite* folder and name it *index.html*.



4. Open *index.html*/in the editor, paste the following text into the file, and save it:

```
<!DOCTYPE html>
<html>
<body>
<h1>Hello World!</h1>
</body>
</html>
```

5. Create the default error file and name it *404.html*.

6. Open *404.html*/in the editor, paste the following text into the file, and save it:

```
<!DOCTYPE html>
<html>
<body>
<h1>404</h1>
</body>
</html>
```

7. Right-click under the *mywebsite* folder in the **Explorer** panel and select **Deploy to Static Website...** to deploy your website. You will be prompted to log in to Azure to retrieve a list of subscriptions.

8. Select the subscription containing the storage account for which you enabled static website hosting. Next, select the storage account when prompted.

Visual Studio Code will now upload your files to your web endpoint, and show the success status bar. Launch the website to view it in Azure.

You've successfully completed the tutorial and deployed a static website to Azure.

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## Next steps

In this tutorial, you learned how to configure your Azure Storage account for static website hosting, and how to create and deploy a static website to an Azure endpoint.

Next, learn how to configure a custom domain with your static website.

[Map a custom domain to an Azure Blob Storage endpoint](#)

# Tutorial: Build a highly available application with Blob storage

8/22/2022 • 12 minutes to read • [Edit Online](#)

This tutorial is part one of a series. In it, you learn how to make your application data highly available in Azure.

When you've completed this tutorial, you will have a console application that uploads and retrieves a blob from a [read-access geo-zone-redundant](#) (RA-GZRS) storage account.

Geo-redundancy in Azure Storage replicates transactions asynchronously from a primary region to a secondary region that is hundreds of miles away. This replication process guarantees that the data in the secondary region is eventually consistent. The console application uses the [circuit breaker](#) pattern to determine which endpoint to connect to, automatically switching between endpoints as failures and recoveries are simulated.

If you don't have an Azure subscription, [create a free account](#) before you begin.

In part one of the series, you learn how to:

- Create a storage account
- Set the connection string
- Run the console application

## Prerequisites

To complete this tutorial:

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)
- [Python v12 SDK](#)
- [Python v2.1](#)
- [Node.js v12 SDK](#)
- [Node.js v11 SDK](#)

We are currently working to create code snippets reflecting version 12.x of the Azure Storage client libraries. For more information, see [Announcing the Azure Storage v12 Client Libraries](#).

## Sign in to the Azure portal

Sign in to the [Azure portal](#).

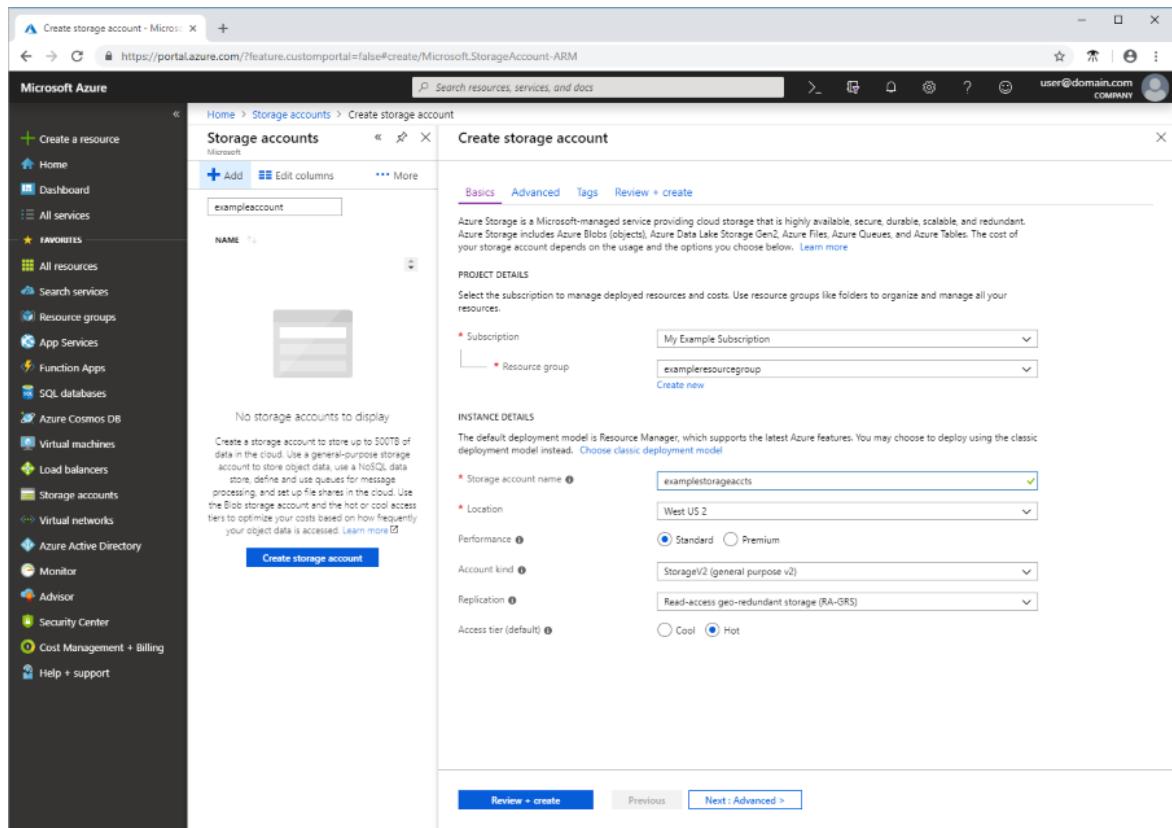
## Create a storage account

A storage account provides a unique namespace to store and access your Azure Storage data objects.

Follow these steps to create a read-access geo-zone-redundant (RA-GZRS) storage account:

1. Select the **Create a resource** button in the Azure portal.
2. Select **Storage account - blob, file, table, queue** from the **New** page.
3. Fill out the storage account form with the following information, as shown in the following image and select **Create**:

Setting	Sample Value	Description
Subscription	<i>My subscription</i>	For details about your subscriptions, see <a href="#">Subscriptions</a> .
ResourceGroup	<i>myResourceGroup</i>	For valid resource group names, see <a href="#">Naming rules and restrictions</a> .
Name	<i>mystorageaccount</i>	A unique name for your storage account.
Location	<i>East US</i>	Choose a location.
Performance	<i>Standard</i>	Standard performance is a good option for the example scenario.
Account kind	<i>StorageV2</i>	Using a general-purpose v2 storage account is recommended. For more information on types of Azure storage accounts, see <a href="#">Storage account overview</a> .
Replication	<i>Read-access geo-zone-redundant storage (RA-GZRS)</i>	The primary region is zone-redundant and is replicated to a secondary region, with read access to the secondary region enabled.
Access tier	<i>Hot</i>	Use the hot tier for frequently-accessed data.



Download the sample

- [.NET v12 SDK](#)

- [.NET v11 SDK](#)
- [Python v12 SDK](#)
- [Python v2.1](#)
- [Node.js v12 SDK](#)
- [Node.js v11 SDK](#)

We are currently working to create code snippets reflecting version 12.x of the Azure Storage client libraries. For more information, see [Announcing the Azure Storage v12 Client Libraries](#).

## Configure the sample

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)
- [Python v12 SDK](#)
- [Python v2.1](#)
- [Node.js v12 SDK](#)
- [Node.js v11 SDK](#)

We are currently working to create code snippets reflecting version 12.x of the Azure Storage client libraries. For more information, see [Announcing the Azure Storage v12 Client Libraries](#).

## Run the console application

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)
- [Python v12 SDK](#)
- [Python v2.1](#)
- [Node.js v12 SDK](#)
- [Node.js v11 SDK](#)

We are currently working to create code snippets reflecting version 12.x of the Azure Storage client libraries. For more information, see [Announcing the Azure Storage v12 Client Libraries](#).

## Understand the sample code

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)
- [Python v12 SDK](#)
- [Python v2.1](#)
- [Node.js v12 SDK](#)
- [Node.js v11 SDK](#)

We are currently working to create code snippets reflecting version 12.x of the Azure Storage client libraries. For more information, see [Announcing the Azure Storage v12 Client Libraries](#).

## Next steps

In part one of the series, you learned about making an application highly available with RA-GZRS storage accounts.

Advance to part two of the series to learn how to simulate a failure and force your application to use the

secondary RA-GZRS endpoint.

[Simulate a failure in reading from the primary region](#)

# Tutorial: Simulate a failure in reading data from the primary region

8/22/2022 • 5 minutes to read • [Edit Online](#)

This tutorial is part two of a series. In it, you learn about the benefits of [read-access geo-zone-redundant storage \(RA-GZRS\)](#) by simulating a failure.

In order to simulate a failure, you can use either [static routing](#) or [Fiddler](#). Both methods will allow you to simulate failure for requests to the primary endpoint of your [read-access geo-redundant \(RA-GZRS\) storage account](#), leading the application to read from the secondary endpoint instead.

If you don't have an Azure subscription, [create a free account](#) before you begin.

In part two of the series, you learn how to:

- Run and pause the application
- Simulate a failure with [an invalid static route](#) or [Fiddler](#)
- Simulate primary endpoint restoration

## Prerequisites

Before you begin this tutorial, complete the previous tutorial: [Make your application data highly available with Azure storage](#).

To simulate a failure with static routing, you will use an elevated command prompt.

To simulate a failure using Fiddler, download and [install Fiddler](#)

## Simulate a failure with an invalid static route

You can create an invalid static route for all requests to the primary endpoint of your [read-access geo-redundant \(RA-GZRS\) storage account](#). In this tutorial, the local host is used as the gateway for routing requests to the storage account. Using the local host as the gateway causes all requests to your storage account primary endpoint to loop back inside the host, which subsequently leads to failure. Follow the following steps to simulate a failure, and primary endpoint restoration with an invalid static route.

### Start and pause the application

Use the instructions in the [previous tutorial](#) to launch the sample and download the test file, confirming that it comes from primary storage. Depending on your target platform, you can then manually pause the sample or wait at a prompt.

### Simulate failure

While the application is paused, open a command prompt on Windows as an administrator or run terminal as root on Linux.

Get information about the storage account primary endpoint domain by entering the following command on a command prompt or terminal, replacing `STORAGEACCOUNTNAME` with the name of your storage account.

```
nslookup STORAGEACCOUNTNAME.blob.core.windows.net
```

Copy to the IP address of your storage account to a text editor for later use.

To get the IP address of your local host, type `ipconfig` on the Windows command prompt, or `ifconfig` on the Linux terminal.

To add a static route for a destination host, type the following command on a Windows command prompt or Linux terminal, replacing `<destination_ip>` with your storage account IP address and `<gateway_ip>` with your local host IP address.

#### Linux

```
route add <destination_ip> gw <gateway_ip>
```

#### Windows

```
route add <destination_ip> <gateway_ip>
```

In the window with the running sample, resume the application or press the appropriate key to download the sample file and confirm that it comes from secondary storage. You can then pause the sample again or wait at the prompt.

#### Simulate primary endpoint restoration

To simulate the primary endpoint becoming functional again, delete the invalid static route from the routing table. This allows all requests to the primary endpoint to be routed through the default gateway. Type the following command on a Windows command prompt or Linux terminal.

#### Linux

```
route del <destination_ip> gw <gateway_ip>
```

#### Windows

```
route delete <destination_ip>
```

You can then resume the application or press the appropriate key to download the sample file again, this time confirming that it once again comes from primary storage.

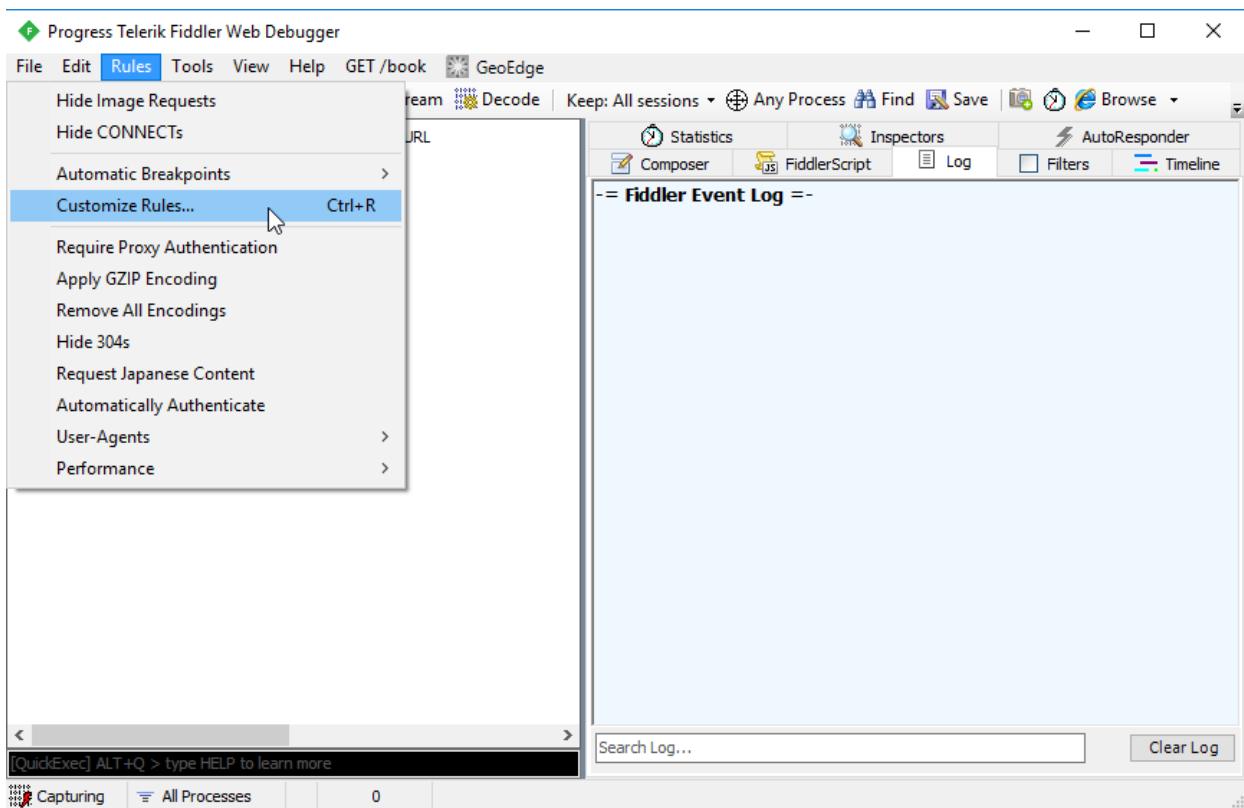
## Simulate a failure with Fiddler

To simulate failure with Fiddler, you inject a failed response for requests to the primary endpoint of your RA-GZRS storage account.

The following sections depict how to simulate a failure and primary endpoint restoration with fiddler.

#### Launch fiddler

Open Fiddler, select **Rules** and **Customize Rules**.



The Fiddler ScriptEditor launches and displays the **SampleRules.js** file. This file is used to customize Fiddler.

Paste the following code sample in the `OnBeforeResponse` function, replacing `STORAGEACCOUNTNAME` with the name of your storage account. Depending on the sample, you may also need to replace `HelloWorld` with the name of the test file (or a prefix such as `sampleFile`) being downloaded. The new code is commented out to ensure that it doesn't run immediately.

Once complete, select **File** and **Save** to save your changes. Leave the ScriptEditor window open for use in the following steps.

```
/*
 // Simulate data center failure
 // After it is successfully downloading the blob, pause the code in the sample,
 // uncomment these lines of script, and save the script.
 // It will intercept the (probably successful) responses and send back a 503 error.
 // When you're ready to stop sending back errors, comment these lines of script out again
 // and save the changes.

 if ((oSession.hostname == "STORAGEACCOUNTNAME.blob.core.windows.net")
 && (oSession.PathAndQuery.Contains("HelloWorld")))
 {
 oSession.responseCode = 503;
 }
 */
```

The screenshot shows the Fiddler ScriptEditor window. The main pane contains a script with several code blocks, some of which are commented out with /\* and \*/. One block checks if the session's hostname is "contosoragrs.blob.core.windows.net" and its path contains "HelloWorld", in which case it sets the response code to 503. The status bar at the bottom left shows the time as 287:39.

```
static function OnBeforeResponse(oSession: Session) {
 if (m_Hide304s && oSession.responseCode == 304) {
 oSession["ui-hide"] = "true";
 }
 /*
 // Simulate data center failure
 // After it is successfully downloading the blob, pause the code in the
 // uncomment these lines of script, and save the script.
 // It will intercept the (probably successful) responses and send back
 // When you're ready to stop sending back errors, comment these lines c
 // and save the changes.

 if ((oSession.hostname == "contosoragrs.blob.core.windows.net")
 && (oSession.PathAndQuery.Contains("HelloWorld"))) {
 oSession.responseCode = 503;
 }
 */
}

/*
<
```

## Start and pause the application

Use the instructions in the [previous tutorial](#) to launch the sample and download the test file, confirming that it comes from primary storage. Depending on your target platform, you can then manually pause the sample or wait at a prompt.

### Simulate failure

While the application is paused, switch back to Fiddler and uncomment the custom rule you saved in the `OnBeforeResponse` function. Be sure to select **File** and **Save** to save your changes so the rule will take effect. This code looks for requests to the RA-GZRS storage account and, if the path contains the name of the sample file, returns a response code of `503 - Service Unavailable`.

In the window with the running sample, resume the application or press the appropriate key to download the sample file and confirm that it comes from secondary storage. You can then pause the sample again or wait at the prompt.

### Simulate primary endpoint restoration

In Fiddler, remove or comment out the custom rule again. Select **File** and **Save** to ensure the rule will no longer be in effect.

In the window with the running sample, resume the application or press the appropriate key to download the sample file and confirm that it comes from primary storage once again. You can then exit the sample.

## Next steps

In part two of the series, you learned about simulating a failure to test read access geo-redundant storage.

To learn more about how RA-GZRS storage works, as well as its associated risks, read the following article:

[Designing HA apps with RA-GZRS](#)

# Tutorial: Add a role assignment condition to restrict access to blobs using the Azure portal (preview)

8/22/2022 • 4 minutes to read • [Edit Online](#)

## IMPORTANT

Azure ABAC and Azure role assignment conditions are currently in preview. This preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

In most cases, a role assignment will grant the permissions you need to Azure resources. However, in some cases you might want to provide more fine-grained access control by adding a role assignment condition.

In this tutorial, you learn how to:

- Add a condition to a role assignment
- Restrict access to blobs based on a blob index tag

## Prerequisites

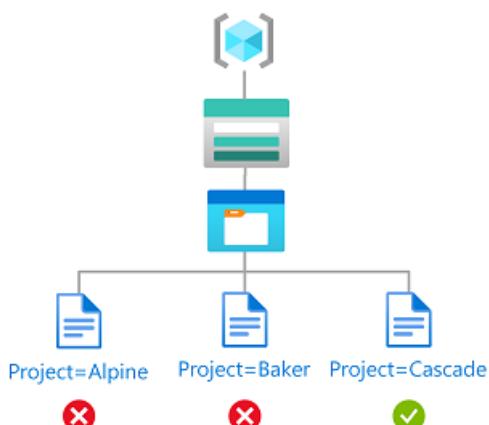
For information about the prerequisites to add or edit role assignment conditions, see [Conditions prerequisites](#).

## Condition

In this tutorial, you restrict access to blobs with a specific tag. For example, you add a condition to a role assignment so that Chandra can only read files with the tag `Project=Cascade`.



If Chandra tries to read a blob without the tag `Project=Cascade`, access is not allowed.



Here is what the condition looks like in code:

```

(
 (
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'}
 AND NOT
 SubOperationMatches{'Blob.List'})
)
 OR
 (
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Project<$key_case_sensitive$>
] StringEqualsIgnoreCase 'Cascade'
)
)

```

## Step 1: Create a user

1. Sign in to the Azure portal as an Owner of a subscription.
2. Click **Azure Active Directory**.
3. Create a user or find an existing user. This tutorial uses Chandra as the example.

## Step 2: Set up storage

1. Create a storage account that is compatible with the blob index tags feature. For more information, see [Manage and find Azure Blob data with blob index tags](#).
2. Create a new container within the storage account and set the Public access level to **Private (no anonymous access)**.
3. In the container, click **Upload** to open the Upload blob pane.
4. Find a text file to upload.
5. Click **Advanced** to expand the pane.
6. In the **Blob index tags** section, add the following blob index tag to the text file.

If you don't see the Blob index tags section and you just registered your subscription, you might need to wait a few minutes for changes to propagate. For more information, see [Use blob index tags to manage and find data on Azure Blob Storage](#).

### NOTE

Blobs also support the ability to store arbitrary user-defined key-value metadata. Although metadata is similar to blob index tags, you must use blob index tags with conditions.

KEY	VALUE
Project	Cascade

The screenshot shows the Azure Storage Blob upload interface. On the left, the 'test-container' blob has been uploaded with the file name 'CascadeFile.txt'. The 'Advanced' settings include 'Azure AD user account' for authentication, 'Block blob' for blob type, and 'Hot (Inferred)' for access tier. A red box highlights the 'Blob index tags' section, which contains a key-value pair: 'Project' with value 'Cascade'. The 'Encryption scope' section shows 'Use existing default container scope' selected.

7. Click the **Upload** button to upload the file.
8. Upload a second text file.
9. Add the following blob index tag to the second text file.

KEY	VALUE
Project	Baker

## Step 3: Assign a storage blob data role

1. Open the resource group.
2. Click **Access control (IAM)**.
3. Click the **Role assignments** tab to view the role assignments at this scope.
4. Click **Add > Add role assignment**.

The screenshot shows the 'Add role assignment' page. The 'Add role assignment' button is highlighted with a red box. The page displays a table with columns: 'Assignments', 'Roles', 'Deny assignments', and 'Classic administrators'. A dropdown menu shows 'this subscription'. At the bottom, there are filters for 'Search by name or e...', 'Type : All', 'Role : All', 'Scope : All scopes', and 'Group by : Role'.

The Add role assignment page opens.

5. On the **Roles** tab, select the **Storage Blob Data Reader** role.

Home > Resource groups > example-group >

## Add role assignment

Got feedback?

**Role** Members Conditions (optional) Review + assign

A role definition is a collection of permissions. You can use the built-in roles or you can create your own custom roles. [Learn more](#)

storage blob Type : All Category : All

Showing 4 of 299 roles

Name ↑↓	Description ↑↓	Type ↑↓	Category ↑↓	Details
Storage Blob Data Contributor	Allows for read, write and delete access to Azure Storage blob containers and data	BuiltinRole	Storage	<a href="#">View</a>
Storage Blob Data Owner	Allows for full access to Azure Storage blob containers and data, including assign...	BuiltinRole	Storage	<a href="#">View</a>
Storage Blob Data Reader	Allows for read access to Azure Storage blob containers and data	BuiltinRole	Storage	<a href="#">View</a>
Storage Blob Delegator	Allows for generation of a user delegation key which can be used to sign SAS toke...	BuiltinRole	Storage	<a href="#">View</a>

Review + assign Previous Next

6. On the **Members** tab, select the user you created earlier.

Home > Resource groups > example-group >

## Add role assignment

Got feedback?

**Role** Members Conditions (optional) Review + assign

**Selected role** Storage Blob Data Reader

**Assign access to**  User, group, or service principal  Managed identity

**Members** + Select members

Name	Object ID	Type
No members selected		

**Description** Optional

Review + assign Previous Next

### Select members

Select ⓘ chandra

Selected members:

Chandra Remove

Select Close

7. (Optional) In the **Description** box, enter **Read access to blobs with the tag Project=Cascade**.

8. Click **Next**.

## Step 4: Add a condition

1. On the **Conditions (optional)** tab, click **Add condition**.

The screenshot shows the 'Add role assignment condition' page. At the top, there's a breadcrumb trail: Home > Resource groups > example-group > Add role assignment >. Below it is a title 'Add role assignment condition' with a 'Preview' link and a close button 'X'. A descriptive text explains that a condition is an additional check for role assignments. The 'Role' is set to 'Storage Blob Data Reader'. The 'Editor type' is 'Visual' (selected). A section titled 'Condition #1' is expanded, showing '1. Add action \*' and '2. Build expression \*'. Under 'Add action', there's a table with columns 'Action Type', 'Action', and 'Operation'. Under 'Build expression', there are buttons for 'Add', 'Delete', 'Group', 'Ungroup', 'Add expression', and '+ Add condition'. At the bottom are 'Save' and 'Discard' buttons.

The Add role assignment condition page appears.

2. In the Add action section, click **Add action**.

The Select an action pane appears. This pane is a filtered list of data actions based on the role assignment that will be the target of your condition.

The screenshot shows the 'Select an action' pane. It includes a note: 'Select the actions you want to allow if the condition is true. If you select multiple actions, there might be fewer attributes to choose from for your condition because the attributes must be available across the selected actions.' A table lists actions under 'Data Action' and 'Description'. The 'Storage Blob Service Blobs' category is expanded, showing 'All read operations.', 'List blobs', and 'Read a blob'. The 'Read a blob' row has a checked checkbox and is highlighted with a red border. At the bottom are 'Select' and 'Discard' buttons.

3. Check the box next to **Read a blob**, then click **Select**.

4. In the Build expression section, click **Add expression**.

The Expression section expands.

5. Specify the following expression settings:

SETTING	VALUE
Attribute source	Resource
Attribute	Blob index tags [Values in key]
Key	Project
Operator	StringEqualsIgnoreCase
Value	Cascade

## 2. Build expression

Build one or more expressions. If the expressions evaluate to true, access is allowed to the selected actions. [Learn more](#)

+ Add expression

1 @Resource[Microsoft.Storage/.../tags:Project<\$key\_case\_sensitive\$>] StringEqualsIgnoreCase 'Cascade'

Attribute source  Resource	Operator  StringEqualsIgnoreCase	Value  Cascade
Attribute  Blob index tags [Values in key]	<input checked="" type="radio"/> Value <input type="radio"/> Attribute	
Key  Project		

Negate this expression

+ Add expression

+ Add condition

Save Discard

## 6. Scroll up to Editor type and click Code.

The condition is displayed as code. You can make changes to the condition in this code editor. To go back to the visual editor, click Visual.

Add role assignment condition ...

Preview

A condition is an additional check that you can optionally add to your role assignment to provide more fine-grained access control. For example, you can add a condition that requires an object to have a specific tag to read the object. [Learn more](#)

**Role** Storage Blob Data Reader

**Editor type**  Visual  Code

```

1 (
2 (
3 !(ActionMatches['Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read']
4 AND NOT SubOperationMatches['Blob.List'])
5)
6 OR
7 (
8 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/
9 tags:Project<$key_case_sensitive$>] StringEqualsIgnoreCase 'Cascade'
10)
11)

```

Save Discard

7. Click **Save** to add the condition and return the Add role assignment page.
8. Click **Next**.
9. On the **Review + assign** tab, click **Review + assign** to assign the role with a condition.

After a few moments, the security principal is assigned the role at the selected scope.

Name	Type	Role	Scope	Condition
Storage Blob Data Reader	User	Storage Blob Data Reader	This resource	
Chandra				

## Step 5: Assign Reader role

- Repeat the previous steps to assign the **Reader** role to the user you created earlier at resource group scope.

### NOTE

You typically don't need to assign the Reader role. However, this is done so that you can test the condition using the Azure portal.

## Step 6: Test the condition

1. In a new window, open the [Azure portal](#).
2. Sign in as the user you created earlier.
3. Open the storage account and container you created.
4. Ensure that the authentication method is set to **Azure AD User Account** and not **Access key**.

The screenshot shows the Azure Storage Explorer interface for a blob container named 'test-container'. The left sidebar lists navigation options like Home, Resource groups, example-group, examplestorage2, and the current container. The main area displays a table of blobs with columns for Name, Modified, Access tier, Blob type, Size, and Lease state. Two blobs are listed: 'BakerFile.txt' and 'CascadeFile.txt'. Both blobs are of type 'Block blob' and have an 'Inferred' access tier. Their sizes are 5 B and 7 B respectively, and both are in an 'Available' lease state.

Name	Modified	Access tier	Blob type	Size	Lease state
BakerFile.txt	10/23/2021, 10:36:58 PM	Hot (Inferred)	Block blob	5 B	Available
CascadeFile.txt	10/23/2021, 10:35:24 PM	Hot (Inferred)	Block blob	7 B	Available

5. Click the Baker text file.

You should NOT be able to view or download the blob and an authorization failed message should be displayed.

6. Click Cascade text file.

You should be able to view and download the blob.

## Step 7: Clean up resources

1. Remove the role assignment you added.
2. Delete the test storage account you created.
3. Delete the user you created.

## Next steps

- [Example Azure role assignment conditions](#)
- [Actions and attributes for Azure role assignment conditions in Azure Storage \(preview\)](#)
- [Azure role assignment condition format and syntax](#)

# Tutorial: Add a role assignment condition to restrict access to blobs using Azure PowerShell (preview)

8/22/2022 • 5 minutes to read • [Edit Online](#)

## IMPORTANT

Azure ABAC and Azure role assignment conditions are currently in preview. This preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

In most cases, a role assignment will grant the permissions you need to Azure resources. However, in some cases you might want to provide more fine-grained access control by adding a role assignment condition.

In this tutorial, you learn how to:

- Add a condition to a role assignment
- Restrict access to blobs based on a blob index tag

## Prerequisites

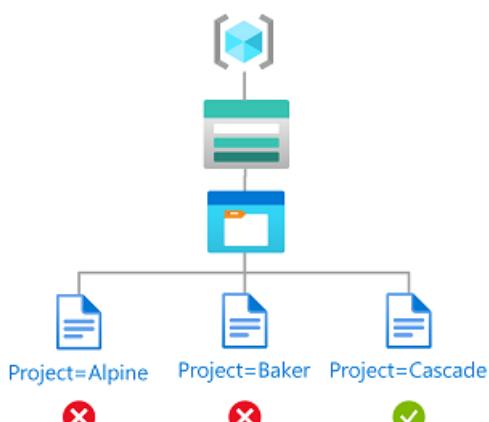
For information about the prerequisites to add or edit role assignment conditions, see [Conditions prerequisites](#).

## Condition

In this tutorial, you restrict access to blobs with a specific tag. For example, you add a condition to a role assignment so that Chandra can only read files with the tag Project=Cascade.



If Chandra tries to read a blob without the tag Project=Cascade, access is not allowed.



Here is what the condition looks like in code:

```
(
 (
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'}
 AND NOT
 SubOperationMatches{'Blob.List'})
)
 OR
 (

 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Project<$key_case_sensitive$>
] StringEquals 'Cascade'
)
)
```

## Step 1: Install prerequisites

1. Open a PowerShell window.
2. Use [Get-InstalledModule](#) to check versions of installed modules.

```
Get-InstalledModule -Name Az
Get-InstalledModule -Name Az.Resources
Get-InstalledModule -Name Az.Storage
```

3. If necessary, use [Install-Module](#) to install the required versions for the [Az](#), [Az.Resources](#), and [Az.Storage](#) modules.

```
Install-Module -Name Az -RequiredVersion 5.5.0
Install-Module -Name Az.Resources -RequiredVersion 3.2.1
Install-Module -Name Az.Storage -RequiredVersion 2.5.2-preview -AllowPrerelease
```

4. Close and reopen PowerShell to refresh session.

## Step 2: Sign in to Azure

1. Use the [Connect-AzAccount](#) command and follow the instructions that appear to sign in to your directory as [User Access Administrator](#) or [Owner](#).

```
Connect-AzAccount
```

2. Use [Get-AzSubscription](#) to list all of your subscriptions.

```
Get-AzSubscription
```

3. Determine the subscription ID and initialize the variable.

```
$subscriptionId = "<subscriptionId>"
```

4. Set the subscription as the active subscription.

```
$context = Get-AzSubscription -SubscriptionId $subscriptionId
Set-AzContext $context
```

## Step 3: Create a user

1. Use [New-AzureADUser](#) to create a user or find an existing user. This tutorial uses Chandra as the example.
2. Initialize the variable for the object ID of the user.

```
$userObjectId = "<userObjectId>"
```

## Step 4: Set up storage

1. Use [New-AzStorageAccount](#) to create a storage account that is compatible with the blob index feature.  
For more information, see [Manage and find Azure Blob data with blob index tags \(preview\)](#).
2. Use [New-AzStorageContainer](#) to create a new blob container within the storage account and set the Public access level to **Private (no anonymous access)**.
3. Use [Set-AzStorageBlobContent](#) to upload a text file to the container.
4. Add the following blob index tag to the text file. For more information, see [Use blob index tags \(preview\) to manage and find data on Azure Blob Storage](#).

### NOTE

Blobs also support the ability to store arbitrary user-defined key-value metadata. Although metadata is similar to blob index tags, you must use blob index tags with conditions.

KEY	VALUE
Project	Cascade

5. Upload a second text file to the container.
6. Add the following blob index tag to the second text file.

KEY	VALUE
Project	Baker

7. Initialize the following variables with the names you used.

```
$resourceGroup = "<resourceGroup>"
$storageAccountName = "<storageAccountName>"
$containerName = "<containerName>"
$blobNameCascade = "<blobNameCascade>"
$blobNameBaker = "<blobNameBaker>"
```

## Step 5: Assign a role with a condition

1. Initialize the [Storage Blob Data Reader](#) role variables.

```
$roleDefinitionName = "Storage Blob Data Reader"
$roleDefinitionId = "2a2b9908-6ea1-4ae2-8e65-a410df84e7d1"
```

2. Initialize the scope for the resource group.

```
$scope = "/subscriptions/$subscriptionId/resourceGroups/$resourceGroup"
```

3. Initialize the condition.

```
$condition = "((!
(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'}) AND NOT
SubOperationMatches{'Blob.List'})) OR
(@Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Project<`$key_case_se
nsitive`$>] StringEquals 'Cascade'))"
```

In PowerShell, if your condition includes a dollar sign (\$), you must prefix it with a backtick (`). For example, this condition uses dollar signs to delineate the tag key name.

4. Initialize the condition version and description.

```
$conditionVersion = "2.0"
$description = "Read access to blobs with the tag Project=Cascade"
```

5. Use [New-AzRoleAssignment](#) to assign the **Storage Blob Data Reader** role with a condition to the user at a resource group scope.

```
New-AzRoleAssignment -ObjectId $userObjectId -Scope $scope -RoleDefinitionId $roleDefinitionId -
Description $description -Condition $condition -ConditionVersion $conditionVersion
```

Here's an example of the output:

```
RoleAssignmentId : /subscriptions/<subscriptionId>/resourceGroups/<resourceGroup>/providers/Microso
 ft.Authorization/roleAssignments/<roleAssignmentId>
Scope : /subscriptions/<subscriptionId>/resourceGroups/<resourceGroup>
DisplayName : Chandra
SignInName : chandra@contoso.com
RoleDefinitionName : Storage Blob Data Reader
RoleDefinitionId : 2a2b9908-6ea1-4ae2-8e65-a410df84e7d1
ObjectId : <userObjectId>
ObjectType : User
CanDelegate : False
Description : Read access to blobs with the tag Project=Cascade
ConditionVersion : 2.0
Condition : ((!
(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'}) AND NOT
SubOperationMatches{'Blob.List'})) OR
(@Resource[Microsoft.Storage/storageAccounts/blobServices/co
ntainers/blobs/tags:Project<$key_case_sensitive$>] StringEquals 'Cascade'))
```

## Step 6: (Optional) View the condition in the Azure portal

1. In the Azure portal, open the resource group.
2. Click **Access control (IAM)**.
3. On the Role assignments tab, find the role assignment.
4. In the **Condition** column, click **View/Edit** to view the condition.

Home > Resource groups > example-group >

### Add role assignment condition

Got feedback?

**1. Add action \***

Click Add action to select the actions you want to allow if the condition is true. [Learn more](#)

Action Type	Action	Operation
Data Action	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read	Read content from a blob with tag conditions ⓘ

Select actions

**2. Build expression \***

Build one or more expressions for the action you selected. If the expressions evaluate to true, access is allowed to the selected action. [Learn more](#)

1 @Resource[Microsoft.Storage/.../tags:Project<\$key\_case\_sensitive\$>] StringEquals 'Cascade'

Attribute source ⓘ	Resource	Operator ⓘ	StringEquals	Value ⓘ	Cascade
Attribute ⓘ	Blob index tags [Values in key]				
Key ⓘ	Project				

**Save** **Discard** **Delete condition**

## Step 7: Test the condition

1. Open a new PowerShell window.
2. Use [Connect-AzAccount](#) to sign in as Chandra.

```
Connect-AzAccount
```

3. Initialize the following variables with the names you used.

```
$storageAccountName = "<storageAccountName>"
$containerName = "<containerName>"
$blobNameBaker = "<blobNameBaker>"
$blobNameCascade = "<blobNameCascade>"
```

4. Use [New-AzStorageContext](#) to create a specific context to access your storage account more easily.

```
$bearerCtx = New-AzStorageContext -StorageAccountName $storageAccountName
```

5. Use [Get-AzStorageBlob](#) to try to read the file for the Baker project.

```
Get-AzStorageBlob -Container $containerName -Blob $blobNameBaker -Context $bearerCtx
```

Here's an example of the output. Notice that you can't read the file because of the condition you added.

```

Get-AzStorageBlob : This request is not authorized to perform this operation using this permission.
HTTP Status Code:
403 - HTTP Error Message: This request is not authorized to perform this operation using this
permission.
ErrorCode: AuthorizationPermissionMismatch
ErrorMessage: This request is not authorized to perform this operation using this permission.
RequestId: <requestId>
Time: Sat, 24 Apr 2021 13:26:25 GMT
At line:1 char:1
+ Get-AzStorageBlob -Container $containerName -Blob $blobNameBaker -Con ...
+ ~~~~~
+ CategoryInfo : CloseError: (:) [Get-AzStorageBlob], StorageException
+ FullyQualifiedErrorId :
StorageException,Microsoft.WindowsAzure.Commands.Storage.Blob.Cmdlet.GetAzureStorageBlob
Command

```

## 6. Read the file for the Cascade project.

```
Get-AzStorageBlob -Container $containerName -Blob $blobNameCascade -Context $bearerCtx
```

Here's an example of the output. Notice that you can read the file because it has the tag Project=Cascade.

AccountName: <storageAccountName>, ContainerName: <containerName>				
Name	BlobType	Length	ContentType	LastModified
AccessTier	SnapshotT			
ime				
-----	-----	-----	-----	-----
-----	-----	-----	-----	-----
CascadeFile.txt	BlockBlob	7	text/plain	2021-04-24 05:35:24Z
Hot				

## Step 8: (Optional) Edit the condition

- In the other PowerShell window, use [Get-AzRoleAssignment](#) to get the role assignment you added.

```
$testRa = Get-AzRoleAssignment -Scope $scope -RoleDefinitionName $roleDefinitionName -ObjectId
$userObjectId
```

- Edit the condition.

```
$condition = "(!
(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'}) AND NOT
SubOperationMatches{'Blob.List'})) OR
(@Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Project<$key_case_se
nsitive`$>] StringEquals 'Cascade' OR
@Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Project<$key_case_se
nsitive`$>] StringEquals 'Baker'))"
```

- Initialize the condition and description.

```
$testRa.Condition = $condition
$testRa.Description = "Read access to blobs with the tag Project=Cascade or Project=Baker"
```

- Use [Set-AzRoleAssignment](#) to update the condition for the role assignment.

```
Set-AzRoleAssignment -InputObject $testRa -PassThru
```

Here's an example of the output:

```
RoleAssignmentId : /subscriptions/<subscriptionId>/resourceGroups/<resourceGroup>/providers/Microsoft.Authorization/roleAssignments/<roleAssignmentId>
Scope : /subscriptions/<subscriptionId>/resourceGroups/<resourceGroup>
DisplayName : Chandra
SignInName : chandra@contoso.com
RoleDefinitionName : Storage Blob Data Reader
RoleDefinitionId : 2a2b9908-6ea1-4ae2-8e65-a410df84e7d1
ObjectId : <userObjectId>
ObjectType : User
CanDelegate : False
Description : Read access to blobs with the tag Project=Cascade or Project=Baker
ConditionVersion : 2.0
Condition : ((!
(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'}) AND NOT
SubOperationMatches{'Blob.List'})) OR
(@Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Project<$key_case_sensitive$>] StringEquals 'Cascade' OR
@Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Project<$key_case_sensitive$>]
StringEquals 'Baker'))
```

## Step 9: Clean up resources

1. Use [Remove-AzRoleAssignment](#) to remove the role assignment and condition you added.

```
Remove-AzRoleAssignment -ObjectId $userObjectId -RoleDefinitionName $roleDefinitionName -
ResourceGroupName $resourceGroup
```

2. Delete the storage account you created.
3. Delete the user you created.

## Next steps

- [Example Azure role assignment conditions](#)
- [Actions and attributes for Azure role assignment conditions in Azure Storage \(preview\)](#)
- [Azure role assignment condition format and syntax](#)

# Tutorial: Add a role assignment condition to restrict access to blobs using Azure CLI (preview)

8/22/2022 • 6 minutes to read • [Edit Online](#)

## IMPORTANT

Azure ABAC and Azure role assignment conditions are currently in preview. This preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

In most cases, a role assignment will grant the permissions you need to Azure resources. However, in some cases you might want to provide more fine-grained access control by adding a role assignment condition.

In this tutorial, you learn how to:

- Add a condition to a role assignment
- Restrict access to blobs based on a blob index tag

## Prerequisites

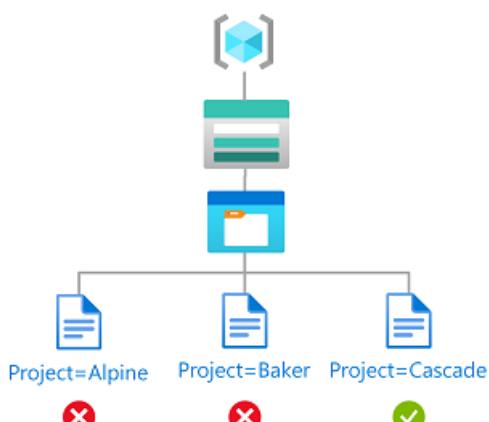
For information about the prerequisites to add or edit role assignment conditions, see [Conditions prerequisites](#).

## Condition

In this tutorial, you restrict access to blobs with a specific tag. For example, you add a condition to a role assignment so that Chandra can only read files with the tag Project=Cascade.



If Chandra tries to read a blob without the tag Project=Cascade, access is not allowed.



Here is what the condition looks like in code:

```
(
 (
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'}
 AND NOT
 SubOperationMatches{'Blob.List'})
)
 OR
 (

 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Project<$key_case_sensitive$>
] StringEquals 'Cascade'
)
)
```

## Step 1: Sign in to Azure

1. Use the [az login](#) command and follow the instructions that appear to sign in to your directory as [User Access Administrator](#) or [Owner](#).

```
az login
```

2. Use [az account show](#) to get the ID of your subscriptions.

```
az account show
```

3. Determine the subscription ID and initialize the variable.

```
subscriptionId=<subscriptionId>
```

## Step 2: Create a user

1. Use [az ad user create](#) to create a user or find an existing user. This tutorial uses Chandra as the example.
2. Initialize the variable for the object ID of the user.

```
userObjectId=<userObjectId>
```

## Step 3: Set up storage

You can authorize access to Blob storage from the Azure CLI either with Azure AD credentials or by using the storage account access key. This article shows how to authorize Blob storage operations using Azure AD. For more information, see [Quickstart: Create, download, and list blobs with Azure CLI](#)

1. Use [az storage account](#) to create a storage account that is compatible with the blob index feature. For more information, see [Manage and find Azure Blob data with blob index tags \(preview\)](#).
2. Use [az storage container](#) to create a new blob container within the storage account and set the Public access level to **Private (no anonymous access)**.
3. Use [az storage blob upload](#) to upload a text file to the container.
4. Add the following blob index tag to the text file. For more information, see [Use blob index tags \(preview\) to manage and find data on Azure Blob Storage](#).

#### NOTE

Blobs also support the ability to store arbitrary user-defined key-value metadata. Although metadata is similar to blob index tags, you must use blob index tags with conditions.

KEY	VALUE
Project	Cascade

5. Upload a second text file to the container.
6. Add the following blob index tag to the second text file.

KEY	VALUE
Project	Baker

7. Initialize the following variables with the names you used.

```
resourceGroup=<resourceGroup>
storageAccountName=<storageAccountName>
containerName=<containerName>
blobNameCascade=<blobNameCascade>
blobNameBaker=<blobNameBaker>
```

## Step 4: Assign a role with a condition

1. Initialize the [Storage Blob Data Reader](#) role variables.

```
roleDefinitionName="Storage Blob Data Reader"
roleDefinitionId="2a2b9908-6ea1-4ae2-8e65-a410df84e7d1"
```

2. Initialize the scope for the resource group.

```
scope="/subscriptions/$subscriptionId/resourceGroups/$resourceGroup"
```

3. Initialize the condition.

```
condition="((!(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'}
AND NOT SubOperationMatches{'Blob.List'})) OR
(@Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Project<\$key_case_sensitive\$\$] StringEquals 'Cascade'))"
```

In Bash, if history expansion is enabled, you might see the message `bash: !: event not found` because of the exclamation point (!). In this case, you can disable history expansion with the command `set +H`. To re-enable history expansion, use `set -H`.

In Bash, a dollar sign (\$) has special meaning for expansion. If your condition includes a dollar sign (\$), you might need to prefix it with a backslash (\). For example, this condition uses dollar signs to delineate the tag key name. For more information about rules for quotation marks in Bash, see [Double Quotes](#).

4. Initialize the condition version and description.

```
conditionVersion="2.0"
description="Read access to blobs with the tag Project=Cascade"
```

5. Use [az role assignment create](#) to assign the [Storage Blob Data Reader](#) role with a condition to the user at a resource group scope.

```
az role assignment create --assignee-object-id $userObjectId --scope $scope --role $roleDefinitionId
--description "$description" --condition "$condition" --condition-version $conditionVersion
```

Here's an example of the output:

```
{
 "canDelegate": null,
 "condition": "((!
(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'} AND NOT
SubOperationMatches{'Blob.List'})) OR
(@Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Project<$key_case_sensitive$>] StringEquals 'Cascade'))",
 "conditionVersion": "2.0",
 "description": "Read access to blobs with the tag Project=Cascade",
 "id": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroup}/providers/Microsoft.Authorization/roleAssignments/{roleAssignmentId}",
 "name": "{roleAssignmentId}",
 "principalId": "{userObjectId}",
 "principalType": "User",
 "resourceGroup": "{resourceGroup}",
 "roleDefinitionId": "/subscriptions/{subscriptionId}/providers/Microsoft.Authorization/roleDefinitions/2a2b9908-6ea1-4ae2-8e65-a410df84e7d1",
 "scope": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroup}",
 "type": "Microsoft.Authorization/roleAssignments"
}
```

## Step 5: (Optional) View the condition in the Azure portal

1. In the Azure portal, open the resource group.
2. Click **Access control (IAM)**.
3. On the Role assignments tab, find the role assignment.
4. In the **Condition** column, click **View/Edit** to view the condition.

Home > Resource groups > example-group >

## Add role assignment condition

Got feedback?

**1. Add action \***

Click Add action to select the actions you want to allow if the condition is true. [Learn more](#)

Action Type	Action	Operation
Data Action	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read	Read content from a blob with tag conditions ⓘ

Select actions

**2. Build expression \***

Build one or more expressions for the action you selected. If the expressions evaluate to true, access is allowed to the selected action. [Learn more](#)

1 @Resource[Microsoft.Storage/.../tags:Project<\$key\_case\_sensitive\$>] StringEquals 'Cascade'

Attribute source ⓘ	Resource	Operator ⓘ	StringEquals	Value ⓘ	Cascade
Attribute ⓘ	Blob index tags [Values in key]				
Key ⓘ	Project				

**Save** **Discard** **Delete condition**

## Step 6: Test the condition

1. Open a new command window.

2. Use [az login](#) to sign in as Chandra.

```
az login
```

3. Initialize the following variables with the names you used.

```
storageAccountName=<storageAccountName>
containerName=<containerName>
blobNameBaker=<blobNameBaker>
blobNameCascade=<blobNameCascade>
```

4. Use [az storage blob show](#) to try to read the properties of the file for the Baker project.

```
az storage blob show --account-name $storageAccountName --container-name $containerName --name
$blobNameBaker --auth-mode login
```

Here's an example of the output. Notice that you can't read the file because of the condition you added.

```
You do not have the required permissions needed to perform this operation.
Depending on your operation, you may need to be assigned one of the following roles:
"Storage Blob Data Contributor"
"Storage Blob Data Reader"
"Storage Queue Data Contributor"
"Storage Queue Data Reader"

If you want to use the old authentication method and allow querying for the right account key, please
use the "--auth-mode" parameter and "key" value.
```

5. Read the properties of the file for the Cascade project.

```
az storage blob show --account-name $storageAccountName --container-name $containerName --name
$blobNameCascade --auth-mode login
```

Here's an example of the output. Notice that you can read the properties of the file because it has the tag Project=Cascade.

```
{
 "container": "<containerName>",
 "content": "",
 "deleted": false,
 "encryptedMetadata": null,
 "encryptionKeySha256": null,
 "encryptionScope": null,
 "isAppendBlobSealed": null,
 "isCurrentVersion": null,
 "lastAccessedOn": null,
 "metadata": {},
 "name": "<blobNameCascade>",
 "objectReplicationDestinationPolicy": null,
 "objectReplicationSourceProperties": [],
 "properties": {
 "appendBlobCommittedBlockCount": null,
 "blobTier": "Hot",
 "blobTierChangeTime": null,
 "blobTierInferred": true,
 "blobType": "BlockBlob",
 "contentLength": 7,
 "contentRange": null,

 ...

 }
}
```

## Step 7: (Optional) Edit the condition

1. In the other command window, use [az role assignment list](#) to get the role assignment you added.

```
az role assignment list --assignee $userObjectId --resource-group $resourceGroup
```

The output will be similar to the following:

```
[
 {
 "canDelegate": null,
 "condition": "(!
(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'}) AND NOT
SubOperationMatches{'Blob.List'})) OR
(@Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Project<$key_case_sensitive$>] StringEquals 'Cascade'))",
 "conditionVersion": "2.0",
 "description": "Read access to blobs with the tag Project=Cascade",
 "id":
"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroup}/providers/Microsoft.Authorization/roleAssignments/{roleAssignmentId}",
 "name": "{roleAssignmentId}",
 "principalId": "{userObjectId}",
 "principalName": "chandra@contoso.com",
 "principalType": "User",
 "resourceGroup": "{resourceGroup}",
 "roleDefinitionId":
"/subscriptions/{subscriptionId}/providers/Microsoft.Authorization/roleDefinitions/2a2b9908-6ea1-4ae2-8e65-a410df84e7d1",
 "roleDefinitionName": "Storage Blob Data Reader",
 "scope": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroup}",
 "type": "Microsoft.Authorization/roleAssignments"
 }
]
```

2. Create a JSON file with the following format and update the `condition` and `description` properties.

```
{
 "canDelegate": null,
 "condition": "(!
(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'}) AND NOT
SubOperationMatches{'Blob.List'})) OR
(@Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Project<$key_case_sensitive$>] StringEquals 'Cascade' OR
@Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Project<$key_case_sensitive$>] StringEquals 'Baker'))",
 "conditionVersion": "2.0",
 "description": "Read access to blobs with the tag Project=Cascade or Project=Baker",
 "id":
"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroup}/providers/Microsoft.Authorization/roleAssignments/{roleAssignmentId}",
 "name": "{roleAssignmentId}",
 "principalId": "{userObjectId}",
 "principalName": "chandra@contoso.com",
 "principalType": "User",
 "resourceGroup": "{resourceGroup}",
 "roleDefinitionId":
"/subscriptions/{subscriptionId}/providers/Microsoft.Authorization/roleDefinitions/2a2b9908-6ea1-4ae2-8e65-a410df84e7d1",
 "roleDefinitionName": "Storage Blob Data Reader",
 "scope": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroup}",
 "type": "Microsoft.Authorization/roleAssignments"
}
```

3. Use [az role assignment update](#) to update the condition for the role assignment.

```
az role assignment update --role-assignment "./path/roleassignment.json"
```

## Step 8: Clean up resources

1. Use [az role assignment delete](#) to remove the role assignment and condition you added.

```
az role assignment delete --assignee $userObjectId --role "$roleDefinitionName" --resource-group
$resourceGroup
```

2. Delete the storage account you created.

3. Delete the user you created.

## Next steps

- [Example Azure role assignment conditions](#)
- [Actions and attributes for Azure role assignment conditions in Azure Storage \(preview\)](#)
- [Azure role assignment condition format and syntax](#)

# Azure Storage samples using v12 .NET client libraries

8/22/2022 • 2 minutes to read • [Edit Online](#)

The following table provides an overview of our samples repository and the scenarios covered in each sample. Click on the links to view the corresponding sample code in GitHub.

## NOTE

These samples use the latest Azure Storage .NET v12 library. For legacy v11 code, see [Azure Blob Storage Samples for .NET](#) in the GitHub repository.

## Blob samples

### Authentication

[Authenticate using a connection string](#)

[Authenticate using a shared key credential](#)

[Authenticate with Azure Identity](#)

[Authenticate using an Active Directory token](#)

[Anonymously access a public blob](#)

### Batching

[Delete several blobs in one request](#)

[Set several blob access tiers in one request](#)

[Fine-grained control in a batch request](#)

[Catch errors from a failed sub-operation](#)

### Blob

[Upload a file to a blob](#)

[Download a blob to a file](#)

[Download an image](#)

[List all blobs in a container](#)

### Troubleshooting

[Trigger a recoverable error using a container client](#)

## Data Lake Storage Gen2 samples

### Authentication

[Anonymously access a public file](#)

[Authenticate using a shared key credential](#)

[Authenticate using a shared access signature \(SAS\)](#)

[Authenticate using an Active Directory token](#)

### **File system**

[Create a file using a file system client](#)

[Get properties on a file and a directory](#)

[Rename a file and a directory](#)

### **Directory**

[Create a directory](#)

[Create a file using a directory client](#)

[List directories](#)

[Traverse files and directories](#)

### **File**

[Upload a file](#)

[Upload by appending to a file](#)

[Download a file](#)

[Set and get a file access control list](#)

[Set and get permissions of a file](#)

### **Troubleshooting**

[Trigger a recoverable error](#)

## Azure Files samples

### **Authentication**

[Authenticate using a connection string](#)

[Authenticate using a shared key credential](#)

[Authenticate using a shared access signature \(SAS\)\)](#)

### **File shares**

[Create a share and upload a file](#)

[Download a file](#)

[Traverse files and directories](#)

### **Troubleshooting**

[Trigger a recoverable error using a share client](#)

## Queue samples

### **Authentication**

[Authenticate using Azure Active Directory](#)

[Authenticate using a connection string](#)

[Authenticate using a shared key credential](#)

[Authenticate using a shared access signature \(SAS\)\)](#)

[Authenticate using an Active Directory token](#)

## **Queue**

[Create a queue and add a message](#)

## **Message**

[Receive and process messages](#)

[Peek at messages](#)

[Receive messages and update visibility timeout](#)

## **Troubleshooting**

[Trigger a recoverable error using a queue client](#)

# Table samples (v11)

[Create table](#)

[Delete entity/table](#)

[Insert/merge/replace entity](#)

[Query entities](#)

[Query tables](#)

[Table ACL/properties](#)

[Update entity](#)

# Azure code sample libraries

To view the complete .NET sample libraries, go to:

- [Azure blob code samples](#)
- [Azure Data Lake code samples](#)
- [Azure Files code samples](#)
- [Azure queue code samples](#)

You can browse and clone the GitHub repository for each library.

# Getting started guides

Check out the following guides if you are looking for instructions on how to install and get started with the Azure Storage Client Libraries.

- [Getting Started with Azure Blob Service in .NET](#)
- [Getting Started with Azure Queue Service in .NET](#)
- [Getting Started with Azure Table Service in .NET](#)
- [Getting Started with Azure File Service in .NET](#)

# Next steps

For information on samples for other languages:

- Java: [Azure Storage samples using Java](#)
- Python: [Azure Storage samples using Python](#)
- JavaScript/Node.js: [Azure Storage samples using JavaScript](#)
- C++: [Azure Storage samples using C++](#)
- All other languages: [Azure Storage samples](#)

# Azure Storage samples using v12 Java client libraries

8/22/2022 • 2 minutes to read • [Edit Online](#)

The following table provides an overview of our samples repository and the scenarios covered in each sample. Click on the links to view the corresponding sample code in GitHub.

## NOTE

These samples use the latest Azure Storage Java v12 library. For legacy v8 code, see [Getting Started with Azure Blob Service in Java](#) in the GitHub repository.

## Blob samples

### Authentication

[Authenticate using a shared key credential](#)

[Authenticate using Azure Identity](#)

### Blob service

[Create a blob service client](#)

[List containers](#)

[Delete containers](#)

### Batching

[Create a blob batch client](#)

[Bulk delete blobs](#)

[Set access tier on a batch of blobs](#)

### Container

[Create a container client](#)

[Create a container](#)

[List blobs](#)

[Delete a container](#)

### Blob

[Upload a blob](#)

[Download a blob](#)

[Delete a blob](#)

[Upload a blob from a large file](#)

[Download a large blob to a file](#)

### Troubleshooting

[Trigger a recoverable error using a container client](#)

# Data Lake Storage Gen2 samples

## **Data Lake service**

[Create a Data Lake service client](#)

[Create a file system client](#)

## **File system**

[Create a file system](#)

[Create a directory](#)

[Create a file and subdirectory](#)

[Create a file client](#)

[List paths in a file system](#)

[Delete a file system](#)

[List file systems in an Azure storage account](#)

## **Directory**

[Create a directory client](#)

[Create a parent directory](#)

[Create a child directory](#)

[Create a file in a child directory](#)

[Get directory properties](#)

[Delete a child directory](#)

[Delete a parent folder](#)

## **File**

[Create a file using a file client](#)

[Delete a file](#)

[Set access controls on a file](#)

[Get access controls on a file](#)

# Azure File samples

## **Authentication**

[Authenticate using a connection string](#)

## **File service**

[Create file shares](#)

[Get properties](#)

[List shares](#)

[Delete shares](#)

## **File share**

[Create a share client](#)

[Create a share](#)

[Create a share snapshot](#)

[Create a directory using a share client](#)

[Get properties of a share](#)

[Get root directory and list directories](#)

[Delete a share](#)

### **Directory**

[Create a parent directory](#)

[Create a child directory](#)

[Create a file in a child directory](#)

[List directories and files](#)

[Delete a child folder](#)

[Delete a parent folder](#)

### **File**

[Create a file client](#)

[Upload a file](#)

[Download a file](#)

[Get file properties](#)

[Delete a file](#)

## Queue samples

### **Authentication**

[Authenticate using a SAS token](#)

### **Queue service**

[Create a queue](#)

[List queues](#)

[Delete queues](#)

### **Queue**

[Create a queue client](#)

[Add messages to a queue](#)

### **Message**

[Get the count of messages](#)

[Peek at messages](#)

[Receive messages](#)

[Update a message](#)

[Delete the first message](#)

[Clear all messages](#)

[Delete a queue](#)

## Table samples (v11)

[Create table](#)

[Delete entity/table](#)

[Insert/merge/replace entity](#)

[Query entities](#)

[Query tables](#)

[Table ACL/properties](#)

[Update entity](#)

## Azure code sample libraries

To view the complete Java sample libraries, go to:

- [Azure blob code samples](#)
- [Azure Data Lake code samples](#)
- [Azure Files code samples](#)
- [Azure queue code samples](#)

You can browse and clone the GitHub repository for each library.

## Getting started guides

Check out the following guides if you are looking for instructions on how to install and get started with the Azure Storage Client Libraries.

- [Getting Started with Azure Blob Service in Java](#)
- [Getting Started with Azure Queue Service in Java](#)
- [Getting Started with Azure Table Service in Java](#)
- [Getting Started with Azure File Service in Java](#)

## Next steps

For information on samples for other languages:

- .NET: [Azure Storage samples using .NET](#)
- Python: [Azure Storage samples using Python](#)
- JavaScript/Node.js: [Azure Storage samples using JavaScript](#)
- C++: [Azure Storage samples using C++](#)
- All other languages: [Azure Storage samples](#)

# Azure Storage samples using v12 Python client libraries

8/22/2022 • 3 minutes to read • [Edit Online](#)

The following tables provide an overview of our samples repository and the scenarios covered in each sample. Click on the links to view the corresponding sample code in GitHub.

## NOTE

These samples use the latest Azure Storage .NET v12 library. For legacy v2.1 code, see [Azure Storage: Getting Started with Azure Storage in Python](#) in the GitHub repository.

## Blob samples

### Authentication

[Create blob service client using a connection string](#)

[Create container client using a connection string](#)

[Create blob client using a connection string](#)

[Create blob service client using a shared access key](#)

[Create blob client from URL](#)

[Create blob client SAS URL](#)

[Create blob service client using ClientSecretCredential](#)

[Create SAS token](#)

[Create blob service client using Azure Identity](#)

[Create blob snapshot](#)

### Blob service

[Get blob service account info](#)

[Set blob service properties](#)

[Get blob service properties](#)

[Get blob service stats](#)

[Create container using service client](#)

[List containers](#)

[Delete container using service client](#)

[Get container client](#)

[Get blob client](#)

### Container

[Create container client from service](#)

[Create container client using SAS URL](#)

[Create container using container client](#)

[Get container properties](#)

[Delete container using container client](#)

[Acquire lease on container](#)

[Set container metadata](#)

[Set container access policy](#)

[Get container access policy](#)

[Generate SAS token](#)

[Create container client using SAS token](#)

[Upload blob to container](#)

[List blobs in container](#)

[Get blob client](#)

## **Blob**

[Upload a blob](#)

[Download a blob](#)

[Delete blob](#)

[Undelete blob](#)

[Get blob properties](#)

[Delete multiple blobs](#)

[Copy blob from URL](#)

[Abort copy blob from URL](#)

[Acquire lease on blob](#)

# Data Lake Storage Gen2 samples

## **Data Lake service**

[Create Data Lake service client](#)

## **File system**

[Create file system client](#)

[Delete file system](#)

## **Directory**

[Create directory client](#)

[Get directory permissions](#)

[Set directory permissions](#)

[Rename directory](#)

[Get directory properties](#)

[Delete directory](#)

## **File**

[Create file client](#)

[Create file](#)

[Get file permissions](#)

[Set file permissions](#)

[Append data to file](#)

[Read data from file](#)

# Azure Files samples

## **Authentication**

[Create share service client from connection string](#)

[Create share service client from account and access key](#)

[Generate SAS token](#)

## **File service**

[Set service properties](#)

[Get service properties](#)

[Create shares using file service client](#)

[List shares using file service client](#)

[Delete shares using file service client](#)

## **File share**

[Create share client from connection string](#)

[Get share client](#)

[Create share using file share client](#)

[Create share snapshot](#)

[Delete share using file share client](#)

[Set share quota](#)

[Set share metadata](#)

[Get share properties](#)

## **Directory**

[Create directory](#)

[Upload file to directory](#)

[Delete file from directory](#)

[Delete directory](#)

[Create subdirectory](#)

[List directories and files](#)

[Delete subdirectory](#)

[Get subdirectory client](#)

[List files in directory](#)

## **File**

[Create file client](#)

[Create file](#)

[Upload file](#)

[Download file](#)

[Delete file](#)

[Copy file from URL](#)

# Queue samples

## **Authentication**

[Authenticate using connection string](#)

[Create queue service client token](#)

[Create queue client from connection string](#)

[Generate queue client SAS token](#)

## **Queue service**

[Create queue service client](#)

[Set queue service properties](#)

[Get queue service properties](#)

[Create queue using service client](#)

[Delete queue using service client](#)

## **Queue**

[Create queue client](#)

[Set queue metadata](#)

[Get queue properties](#)

[Create queue using queue client](#)

[Delete queue using queue client](#)

[List queues](#)

[Get queue client](#)

## **Message**

[Send messages](#)

[Receive messages](#)

[Peek message](#)

[Update message](#)

[Delete message](#)

[Clear messages](#)

[Set message access policy](#)

## Table samples (SDK v2.1)

[Create table](#)

[Delete entity/table](#)

[Insert/merge/replace entity](#)

[Query entities](#)

[Query tables](#)

[Table ACL/properties](#)

[Update entity](#)

## Azure code sample libraries

To view the complete Python sample libraries, go to:

- [Azure blob code samples](#)
- [Azure Data Lake code samples](#)
- [Azure Files code samples](#)
- [Azure queue code samples](#)

You can browse and clone the GitHub repository for each library.

## Getting started guides

Check out the following guides if you are looking for instructions on how to install and get started with the Azure Storage client libraries.

- [Getting Started with Azure Blob Service in Python](#)
- [Getting Started with Azure Queue Service in Python](#)
- [Getting Started with Azure Table Service in Python](#)
- [Getting Started with Azure File Service in Python](#)

## Next steps

For information on samples for other languages:

- .NET: [Azure Storage samples using .NET](#)
- Java: [Azure Storage samples using Java](#)
- JavaScript/Node.js: [Azure Storage samples using JavaScript](#)
- C++: [Azure Storage samples using C++](#)

- All other languages: [Azure Storage samples](#)

# Azure Storage samples using v12 JavaScript client libraries

8/22/2022 • 2 minutes to read • [Edit Online](#)

The following tables provide an overview of our samples repository and the scenarios covered in each sample. Click on the links to view the corresponding sample code in GitHub.

## NOTE

These samples use the latest Azure Storage JavaScript v12 library. For legacy v11 code, see [Getting Started with Azure Blob Service in Node.js](#) in the GitHub repository.

## Blob samples

### Authentication

[Authenticate using connection string](#)

[Authenticate using SAS connection string](#)

[Authenticate using shared key credential](#)

[Authenticate using AnonymousCredential](#)

[Authenticate using Azure Active Directory](#)

[Authenticate using a proxy](#)

[Connect using a custom pipeline](#)

### Blob service

[Create blob service client using a SAS URL](#)

### Container

[Create a container](#)

[Create a container using a shared key credential](#)

[List containers](#)

[List containers using an iterator](#)

[List containers by page](#)

[Delete a container](#)

### Blob

[Create a blob](#)

[List blobs](#)

[Download a blob](#)

[List blobs using an iterator](#)

[List blobs by page](#)

[List blobs by hierarchy](#)

[Listing blobs without using await](#)

[Create a blob snapshot](#)

[Download a blob snapshot](#)

[Parallel upload a stream to a blob](#)

[Parallel download block blob](#)

[Set the access tier on a blob](#)

### **Troubleshooting**

[Trigger a recoverable error using a container client](#)

## Data Lake Storage Gen2 samples

[Create a Data Lake service client](#)

[Create a file system](#)

[List file systems](#)

[Create a file](#)

[List paths in a file system](#)

[Download a file](#)

[Delete a file system](#)

## Azure Files samples

### **Authentication**

[Authenticate using a connection string](#)

[Authenticate using a shared key credential](#)

[Authenticate using AnonymousCredential](#)

[Connect using a custom pipeline](#)

[Connect using a proxy](#)

### **Share**

[Create a share](#)

[List shares](#)

[List shares by page](#)

[Delete a share](#)

### **Directory**

[Create a directory](#)

[List files and directories](#)

[List files and directories by page](#)

## **File**

[Parallel upload a file](#)

[Parallel upload a readable stream](#)

[Parallel download a file](#)

[List file handles](#)

[List file handles by page](#)

# Queue samples

## **Authentication**

[Authenticate using a connection string](#)

[Authenticate using a shared key credential](#)

[Authenticate using AnonymousCredential](#)

[Connect using a custom pipeline](#)

[Connect using a proxy](#)

[Authenticate using Azure Active Directory](#)

## **Queue service**

[Create a queue service client](#)

## **Queue**

[Create a new queue](#)

[List queues](#)

[List queues by page](#)

[Delete a queue](#)

## **Message**

[Send a message into a queue](#)

[Peek at messages](#)

[Receive messages](#)

[Delete messages](#)

# Table samples (v11)

[Batch entities](#)

[Create table](#)

[Delete entity/table](#)

[Insert/merge/replace entity](#)

[List tables](#)

[Query entities](#)

[Query tables](#)

[Range query](#)

[Shared Access Signature \(SAS\)](#)

[Table ACL](#)

[Table Cross-Origin Resource Sharing \(CORS\) rules](#)

[Table properties](#)

[Table stats](#)

[Update entity](#)

## Azure code sample libraries

To view the complete JavaScript sample libraries, go to:

- [Azure Blob code samples](#)
- [Azure Data Lake code samples](#)
- [Azure Files code samples](#)
- [Azure Queue code samples](#)

You can browse and clone the GitHub repository for each library.

## Getting started guides

Check out the following guides if you are looking for instructions on how to install and get started with the Azure Storage Client Libraries.

- [Getting Started with Azure Blob Service in JavaScript](#)
- [Getting Started with Azure Queue Service in JavaScript](#)
- [Getting Started with Azure Table Service in JavaScript](#)

## Next steps

For information on samples for other languages:

- .NET: [Azure Storage samples using .NET](#)
- Java: [Azure Storage samples using Java](#)
- Python: [Azure Storage samples using Python](#)
- C++: [Azure Storage samples using C++](#)
- All other languages: [Azure Storage samples](#)

# Azure Storage samples using v12 C++ client libraries

8/22/2022 • 2 minutes to read • [Edit Online](#)

The following table provides an overview of our samples repository and the scenarios covered in each sample. Click on the links to view the corresponding sample code in GitHub.

## NOTE

These samples use the latest Azure Storage C++ v12 library.

## Blob samples

[Authenticate using a connection string](#)

[Create a blob container](#)

[Get a blob client](#)

[Upload a blob](#)

[Set metadata on a blob](#)

[Get blob properties](#)

[Download a blob](#)

## Data Lake Storage Gen2 samples

[Create a service client using a connection string](#)

[Create a file system client using a connection string](#)

[Create a file system](#)

[Create a directory](#)

[Create a file](#)

[Append data to a file](#)

[Flush file data](#)

[Read a file](#)

[List all file systems](#)

[Delete file system](#)

## Azure Files samples

[Create a share client using a connection string](#)

[Create a file share](#)

[Get a file client](#)

[Upload a file](#)

[Set metadata on a file](#)

[Get file properties](#)

[Download a file](#)

## Azure code sample libraries

To view the complete C++ sample libraries, go to:

- [Azure Blob code samples](#)
- [Azure Data Lake code samples](#)
- [Azure Files code samples](#)

You can browse and clone the GitHub repository for each library.

## Getting started guides

Check out the following guides if you are looking for instructions on how to install and get started with the Azure Storage Client Libraries.

- [Quickstart: Azure Blob storage library v12 - C++](#)

## Next steps

For information on samples for other languages:

- .NET: [Azure Storage samples using .NET](#)
- Java: [Azure Storage samples using Java](#)
- Python: [Azure Storage samples using Python](#)
- JavaScript/Node.js: [Azure Storage samples using JavaScript](#)
- All other languages: [Azure Storage samples](#)

# Azure Storage samples

8/22/2022 • 2 minutes to read • [Edit Online](#)

Use the links below to view and download Azure Storage sample code and applications.

## Azure Code Samples library

The [Azure Code Samples](#) library includes samples for Azure Storage that you can download and run locally. The Code Sample Library provides sample code in .zip format. Alternatively, you can browse and clone the GitHub repository for each sample.

## .NET samples

To explore the .NET samples, download the [.NET Storage Client Library](#) from NuGet. The .NET storage client library is also available in the [Azure SDK for .NET](#).

- [Azure Storage samples using .NET](#)

## Java samples

To explore the Java samples, download the [Java Storage Client Library](#).

- [Azure Storage samples using Java](#)

## Python samples

To explore the Python samples, download the [Python Storage Client Library](#).

- [Azure Storage samples using Python](#)

## Node.js samples

To explore the Node.js samples, download the [Node.js Storage Client Library](#).

- [Azure Storage samples using JavaScript/Node.js](#)

## C++ samples

To explore the C++ samples, get the [Azure Storage Client Library for C++](#) from GitHub.

- [Get started with Azure Blobs](#)
- [Get started with Azure Data Lake](#)
- [Get started with Azure Files](#)

## Azure CLI

To explore the Azure CLI samples, first [Install the Azure CLI](#).

- [Get started with the Azure CLI](#)
- [Azure Storage samples using the Azure CLI](#)

## API reference and source code

LANGUAGE	API REFERENCE	SOURCE CODE
.NET	<a href="#">.NET Client Library Reference</a>	Source code for the .NET storage client library
Java	<a href="#">Java Client Library Reference</a>	Source code for the Java storage client library
Python	<a href="#">Python Client Library Reference</a>	Source code for the Python storage client library
Node.js	<a href="#">Node.js Client Library Reference</a>	Source code for the Node.js storage client library
C++	<a href="#">C++ Client Library Reference</a>	Source code for the C++ storage client library
Azure CLI	<a href="#">Azure CLI Library Reference</a>	Source code for the Azure CLI storage client library

## Next steps

The following articles index each of the samples by service (blob, file, queue, table).

- [Azure Storage samples using .NET](#)
- [Azure Storage samples using Java](#)
- [Azure Storage samples using JavaScript](#)
- [Azure Storage samples using Python](#)
- [Azure Storage samples using C++](#)
- [Azure Storage samples using the Azure CLI](#)

# Azure PowerShell samples for Azure Blob storage

8/22/2022 • 2 minutes to read • [Edit Online](#)

The following table includes links to PowerShell script samples that create and manage Azure Storage.

SCRIPT	DESCRIPTION
<b>Storage accounts</b>	
<a href="#">Create a storage account and retrieve/rotate the access keys</a>	Creates an Azure Storage account and retrieves and rotates one of its access keys.
<a href="#">Migrate Blobs across storage accounts using AzCopy on Windows</a>	Migrate blobs across Azure Storage accounts using AzCopy on Windows.
<b>Blob storage</b>	
<a href="#">Calculate the total size of a Blob storage container</a>	Calculates the total size of all the blobs in a container.
<a href="#">Calculate the size of a Blob storage container for billing purposes</a>	Calculates the size of a container in Blob storage for the purpose of estimating billing costs.
<a href="#">Delete containers with a specific prefix</a>	Deletes containers starting with a specified string.

# Azure CLI samples for Azure Blob storage

8/22/2022 • 2 minutes to read • [Edit Online](#)

The following table includes links to Bash scripts built using the Azure CLI that create and manage Azure Storage.

SCRIPT	DESCRIPTION
<b>Storage accounts</b>	
<a href="#">Create a storage account and retrieve/rotate the access keys</a>	Creates an Azure Storage account and retrieves and rotates its access keys.
<b>Blob storage</b>	
<a href="#">Calculate the total size of a Blob storage container</a>	Calculates the total size of all the blobs in a container.
<a href="#">Delete containers with a specific prefix</a>	Deletes containers starting with a specified string.

# Create a storage account and rotate its account access keys

8/22/2022 • 3 minutes to read • [Edit Online](#)

This script creates an Azure Storage account, displays the new storage account's access keys, then renews (rotates) the keys.

If you don't have an [Azure subscription](#), create an [Azure free account](#) before you begin.

## Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Azure Cloud Shell Quickstart - Bash](#).

 [Launch Cloud Shell](#)

- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see [How to run the Azure CLI in a Docker container](#).
  - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).

## Sample script

### Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to <https://shell.azure.com>.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment. Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

### Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an [Azure subscription](#), create an [Azure free account](#) before you begin.

```
subscription="<subscriptionId>" # add subscription here
az account set -s $subscription # ...or use 'az login'
```

For more information, see [set active subscription](#) or [log in interactively](#)

## Run the script

```
Rotate storage account keys

Variables for storage
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-azuresql-rg-$randomIdentifier"
tag="rotate-storage-account-keys"
storage="msdocsstorage$randomIdentifier"

Create a resource group
echo "Creating $resourceGroup in $location..."
az group create --name $resourceGroup --location "$location" --tags $tag

Create a general-purpose standard storage account
echo "Creating $storage..."
az storage account create --name $storage --resource-group $resourceGroup --location "$location" --sku Standard_RAGRS --encryption-services blob

List the storage account access keys
az storage account keys list \
 --resource-group $resourceGroup \
 --account-name $storage

Renew (rotate) the PRIMARY access key
az storage account keys renew \
 --resource-group $resourceGroup \
 --account-name $storage \
 --key primary

Renew (rotate) the SECONDARY access key
az storage account keys renew \
 --resource-group $resourceGroup \
 --account-name $storage \
 --key secondary
```

## Clean up resources

Use the following command to remove the resource group and all resources associated with it using the [az group delete](#) command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

```
az group delete --name $resourceGroup
```

## Sample reference

This script uses the following commands to create the storage account and retrieve and rotate its access keys. Each item in the table links to command-specific documentation.

COMMAND	NOTES
<a href="#">az group create</a>	Creates a resource group in which all resources are stored.
<a href="#">az storage account create</a>	Creates an Azure Storage account in the specified resource group.

COMMAND	NOTES
<code>az storage account keys list</code>	Displays the storage account access keys for the specified account.
<code>az storage account keys renew</code>	Regenerates the primary or secondary storage account access key.

## Next steps

For more information on the Azure CLI, see [Azure CLI documentation](#).

Additional storage CLI script samples can be found in the [Azure CLI samples for Azure Blob storage](#).

# Calculate the size of a Blob storage container

8/22/2022 • 4 minutes to read • [Edit Online](#)

This script calculates the size of a container in Azure Blob storage by totaling the size of the blobs in the container.

If you don't have an [Azure subscription](#), create an [Azure free account](#) before you begin.

## IMPORTANT

This CLI script provides an estimated size for the container and should not be used for billing calculations.

The maximum number of blobs returned with a single listing call is 5000. If you need to return more than 5000 blobs, use a continuation token to request additional sets of results.

## Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Azure Cloud Shell Quickstart - Bash](#).

 [Launch Cloud Shell](#)

- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see [How to run the Azure CLI in a Docker container](#).
  - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).

## Sample script

### Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to <https://shell.azure.com>.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment. Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

### Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't

have an [Azure subscription](#), create an [Azure free account](#) before you begin.

```
subscription=<subscriptionId> # add subscription here
az account set -s $subscription # ...or use 'az login'
```

For more information, see [set active subscription](#) or [log in interactively](#)

**Run the script**

```

Calculate container size

Variables for storage
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-azuresql-rg-$randomIdentifier"
tag="calculate-container-size"
storage="msdocsstorage$randomIdentifier"
container="msdocs-storage-container-$randomIdentifier"

Create a resource group
echo "Creating $resourceGroup in $location..."
az group create --name $resourceGroup --location "$location" --tags $tag

Create storage account
echo "Creating $storage..."
az storage account create --name $storage --resource-group $resourceGroup --location "$location" --sku Standard_LRS

Create a container
echo "Creating $container on $storage..."
key=$(az storage account keys list --account-name $storage --resource-group $resourceGroup -o json --query [0].value | tr -d "'")

az storage container create --name $container --account-key $key --account-name $storage #--public-access container

Create sample files to upload as blobs
for i in `seq 1 3`; do
 echo $randomIdentifier > container_size_sample_file_$i.txt
done

Upload sample files to container
az storage blob upload-batch \
 --pattern "container_size_sample_file_*.txt" \
 --source . \
 --destination $container \
 --account-key $key \
 --account-name $storage

Calculate total size of container. Use the --query parameter to display only
blob contentLength and output it in TSV format so only the values are
returned. Then pipe the results to the paste and bc utilities to total the
size of the blobs in the container. The bc utility is not supported in Cloud Shell.
bytes=`az storage blob list \
 --container-name $container \
 --account-key $key \
 --account-name $storage \
 --query "[*].[properties.contentLength]" \
 --output tsv | paste -s -d+ | bc`

Display total bytes
echo "Total bytes in container: $bytes"

Delete the sample files created by this script
rm container_size_sample_file_*.txt

```

## Clean up resources

Use the following command to remove the resource group and all resources associated with it using the [az group delete](#) command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

```
az group delete --name $resourceGroup
```

## Sample reference

This script uses the following commands to calculate the size of the Blob storage container. Each item in the table links to command-specific documentation.

COMMAND	NOTES
<a href="#">az group create</a>	Creates a resource group in which all resources are stored.
<a href="#">az storage blob upload</a>	Uploads local files to an Azure Blob storage container.
<a href="#">az storage blob list</a>	Lists the blobs in an Azure Blob storage container.

## Next steps

For more information on the Azure CLI, see [Azure CLI documentation](#).

Additional storage CLI script samples can be found in the [Azure CLI samples for Azure Blob storage](#).

# Use an Azure CLI script to delete containers based on container name prefix

8/22/2022 • 3 minutes to read • [Edit Online](#)

This script first creates a few sample containers in Azure Blob storage, then deletes some of the containers based on a prefix in the container name.

If you don't have an [Azure subscription](#), create an [Azure free account](#) before you begin.

## Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Azure Cloud Shell Quickstart - Bash](#).

 [Launch Cloud Shell](#)

- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see [How to run the Azure CLI in a Docker container](#).
  - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).

## Sample script

### Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to <https://shell.azure.com>.

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment. Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

### Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an [Azure subscription](#), create an [Azure free account](#) before you begin.

```
subscription="<subscriptionId>" # add subscription here
az account set -s $subscription # ...or use 'az login'
```

For more information, see [set active subscription](#) or [log in interactively](#)

## Run the script

```
Delete containers by prefix

Variables for storage
let "randomIdentifier=$RANDOM*$RANDOM"
location="East US"
resourceGroup="msdocs-azuresql-rg-$randomIdentifier"
tag="delete-containers-by-prefix"
storage="msdocsstorage$randomIdentifier"
container1="msdocs-test1-storage-container-$randomIdentifier"
container2="msdocs-test2-storage-container-test2-$randomIdentifier"
containerProd="msdocs-prod1-storage-$randomIdentifier"

Create a resource group
echo "Creating $resourceGroup in $location..."
az group create --name $resourceGroup --location "$location" --tags $tag

Create storage account
echo "Creating $storage..."
az storage account create --name $storage --resource-group $resourceGroup --location "$location" --sku Standard_LRS

Create some test containers
echo "Creating $container1 $container2 and $containerProd on $storage..."
key=$(az storage account keys list --account-name $storage --resource-group $resourceGroup -o json --query [0].value | tr -d "'")

az storage container create --name $container1 --account-key $key --account-name $storage #--public-access container
az storage container create --name $container2 --account-key $key --account-name $storage #--public-access container
az storage container create --name $containerProd --account-key $key --account-name $storage #--public-access container

List only the containers with a specific prefix
echo "List container with msdocs-test prefix"
az storage container list --account-key $key --account-name $storage --prefix "msdocs-test" --query "[*].[name]" --output tsv

Delete
echo "Deleting msdocs-test containers..."
for container in `az storage container list --account-key $key --account-name $storage --prefix "msdocs-test" --query "[*].[name]" --output tsv`; do
 az storage container delete --account-key $key --account-name $storage --name $container
done

echo "Remaining containers..."
az storage container list --account-key $key --account-name $storage --output table
```

## Clean up resources

Use the following command to remove the resource group and all resources associated with it using the [az group delete](#) command - unless you have an ongoing need for these resources. Some of these resources may take a while to create, as well as to delete.

```
az group delete --name $resourceGroup
```

## Sample reference

This script uses the following commands to delete containers based on container name prefix. Each item in the table links to command-specific documentation.

COMMAND	NOTES
<a href="#">az group create</a>	Creates a resource group in which all resources are stored.
<a href="#">az storage account create</a>	Creates an Azure Storage account in the specified resource group.
<a href="#">az storage container create</a>	Creates a container in Azure Blob storage.
<a href="#">az storage container list</a>	Lists the containers in an Azure Storage account.
<a href="#">az storage container delete</a>	Deletes containers in an Azure Storage account.

## Next steps

For more information on the Azure CLI, see [Azure CLI documentation](#).

Additional storage CLI script samples can be found in the [Azure CLI samples for Azure Storage](#).

# Azure Resource Graph sample queries for Azure Storage

8/22/2022 • 3 minutes to read • [Edit Online](#)

This page is a collection of [Azure Resource Graph](#) sample queries for Azure Storage. For a complete list of Azure Resource Graph samples, see [Resource Graph samples by Category](#) and [Resource Graph samples by Table](#).

## Sample queries

### Find storage accounts with a specific case-insensitive tag on the resource group

Similar to the 'Find storage accounts with a specific case-sensitive tag on the resource group' query, but when it's necessary to look for a case insensitive tag name and tag value, use `mv-expand` with the `bagexpansion` parameter. This query uses more quota than the original query, so use `mv-expand` only if necessary.

```
Resources
| where type =~ 'microsoft.storage/storageaccounts'
| join kind=inner (
 ResourceContainers
 | where type =~ 'microsoft.resources/subscriptions/resourcegroups'
 | mv-expand bagexpansion=array tags
 | where isnotempty(tags)
 | where tags[0] =~ 'key1' and tags[1] =~ 'value1'
 | project subscriptionId, resourceGroup)
on subscriptionId, resourceGroup
| project-away subscriptionId1, resourceGroup1
```

- [Azure CLI](#)
- [Azure PowerShell](#)
- [Portal](#)

```
az graph query -q "Resources | where type =~ 'microsoft.storage/storageaccounts' | join kind=inner (
 ResourceContainers | where type =~ 'microsoft.resources/subscriptions/resourcegroups' | mv-expand
 bagexpansion=array tags | where isnotempty(tags) | where tags[0] =~ 'key1' and tags[1] =~ 'value1' | project
 subscriptionId, resourceGroup) on subscriptionId, resourceGroup | project-away subscriptionId1,
 resourceGroup1"
```

### Find storage accounts with a specific case-sensitive tag on the resource group

The following query uses an `inner join` to connect storage accounts with resource groups that have a specified case-sensitive tag name and tag value.

```
Resources
| where type =~ 'microsoft.storage/storageaccounts'
| join kind=inner (
 ResourceContainers
 | where type =~ 'microsoft.resources/subscriptions/resourcegroups'
 | where tags['Key1'] =~ 'Value1'
 | project subscriptionId, resourceGroup)
on subscriptionId, resourceGroup
| project-away subscriptionId1, resourceGroup1
```

- [Azure CLI](#)
- [Azure PowerShell](#)
- [Portal](#)

```
az graph query -q "Resources | where type =~ 'microsoft.storage/storageaccounts' | join kind=inner (ResourceContainers | where type =~ 'microsoft.resources/subscriptions/resourcegroups' | where tags['Key1'] =~ 'Value1' | project subscriptionId, resourceGroup) on subscriptionId, resourceGroup | project-away subscriptionId1, resourceGroup1"
```

### List all storage accounts with specific tag value

Combine the filter functionality of the previous example and filter Azure resource type by **type** property. This query also limits our search for specific types of Azure resources with a specific tag name and value.

```
Resources
| where type =~ 'Microsoft.Storage/storageAccounts'
| where tags['tag with a space']=='Custom value'
```

- [Azure CLI](#)
- [Azure PowerShell](#)
- [Portal](#)

```
az graph query -q "Resources | where type =~ 'Microsoft.Storage/storageAccounts' | where tags['tag with a space']=='Custom value'"
```

### Show resources that contain storage

Instead of explicitly defining the type to match, this example query will find any Azure resource that **contains** the word **storage**.

```
Resources
| where type contains 'storage' | distinct type
```

- [Azure CLI](#)
- [Azure PowerShell](#)
- [Portal](#)

```
az graph query -q "Resources | where type contains 'storage' | distinct type"
```

## Next steps

- Learn more about the [query language](#).
- Learn more about how to [explore resources](#).
- See samples of [Starter language queries](#).
- See samples of [Advanced language queries](#).

# Storage account overview

8/22/2022 • 12 minutes to read • [Edit Online](#)

An Azure storage account contains all of your Azure Storage data objects, including blobs, file shares, queues, tables, and disks. The storage account provides a unique namespace for your Azure Storage data that's accessible from anywhere in the world over HTTP or HTTPS. Data in your storage account is durable and highly available, secure, and massively scalable.

To learn how to create an Azure Storage account, see [Create a storage account](#).

## Types of storage accounts

Azure Storage offers several types of storage accounts. Each type supports different features and has its own pricing model.

The following table describes the types of storage accounts recommended by Microsoft for most scenarios. All of these use the [Azure Resource Manager](#) deployment model.

Type of storage account	Supported storage services	Redundancy options	Usage
Standard general-purpose v2	Blob Storage (including Data Lake Storage <sup>1</sup> ), Queue Storage, Table Storage, and Azure Files	Locally redundant storage (LRS) / geo-redundant storage (GRS) / read-access geo-redundant storage (RA-GRS)  Zone-redundant storage (ZRS) / geo-zone-redundant storage (GZRS) / read-access geo-zone-redundant storage (RA-GZRS) <sup>2</sup>	Standard storage account type for blobs, file shares, queues, and tables.  Recommended for most scenarios using Azure Storage. If you want support for network file system (NFS) in Azure Files, use the premium file shares account type.
Premium block blobs <sup>3</sup>	Blob Storage (including Data Lake Storage <sup>1</sup> )	LRS  ZRS <sup>2</sup>	Premium storage account type for block blobs and append blobs.  Recommended for scenarios with high transaction rates or that use smaller objects or require consistently low storage latency. <a href="#">Learn more about example workloads</a> .
Premium file shares <sup>3</sup>	Azure Files	LRS  ZRS <sup>2</sup>	Premium storage account type for file shares only.  Recommended for enterprise or high-performance scale applications. Use this account type if you want a storage account that supports both Server Message Block (SMB) and NFS file shares.

Type of storage account	Supported storage services	Redundancy options	Usage
Premium page blobs <sup>3</sup>	Page blobs only	LRS	Premium storage account type for page blobs only. <a href="#">Learn more about page blobs and sample use cases.</a>

<sup>1</sup> Data Lake Storage is a set of capabilities dedicated to big data analytics, built on Azure Blob Storage. For more information, see [Introduction to Data Lake Storage Gen2](#) and [Create a storage account to use with Data Lake Storage Gen2](#).

<sup>2</sup> ZRS, GZRS, and RA-GZRS are available only for standard general-purpose v2, premium block blobs, and premium file shares accounts in certain regions. For more information, see [Azure Storage redundancy](#).

<sup>3</sup> Premium performance storage accounts use solid-state drives (SSDs) for low latency and high throughput.

Legacy storage accounts are also supported. For more information, see [Legacy storage account types](#).

The service-level agreement (SLA) for Azure Storage accounts is available at [SLA for Storage Accounts](#).

#### NOTE

You can't change a storage account to a different type after it's created. To move your data to a storage account of a different type, you must create a new account and copy the data to the new account.

## Storage account name

When naming your storage account, keep these rules in mind:

- Storage account names must be between 3 and 24 characters in length and may contain numbers and lowercase letters only.
- Your storage account name must be unique within Azure. No two storage accounts can have the same name.

## Storage account endpoints

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

There are two types of service endpoints available for a storage account:

- Standard endpoints (recommended). You can create up to 250 storage accounts per region with standard endpoints in a given subscription.
- Azure DNS zone endpoints (preview). You can create up to 5000 storage accounts per region with Azure DNS zone endpoints in a given subscription.

Within a single subscription, you can create accounts with either standard or Azure DNS Zone endpoints, for a maximum of 5250 accounts per subscription.

#### IMPORTANT

Azure DNS zone endpoints are currently in PREVIEW. See the [Supplemental Terms of Use for Microsoft Azure Previews](#) for legal terms that apply to Azure features that are in beta, preview, or otherwise not yet released into general availability.

You can configure your storage account to use a custom domain for the Blob Storage endpoint. For more

information, see [Configure a custom domain name for your Azure Storage account](#).

## Standard endpoints

A standard service endpoint in Azure Storage includes the protocol (HTTPS is recommended), the storage account name as the subdomain, and a fixed domain that includes the name of the service.

The following table lists the format for the standard endpoints for each of the Azure Storage services.

STORAGE SERVICE	ENDPOINT
Blob Storage	<code>https://&lt;storage-account&gt;.blob.core.windows.net</code>
Static website (Blob Storage)	<code>https://&lt;storage-account&gt;.web.core.windows.net</code>
Data Lake Storage Gen2	<code>https://&lt;storage-account&gt;.dfs.core.windows.net</code>
Azure Files	<code>https://&lt;storage-account&gt;.file.core.windows.net</code>
Queue Storage	<code>https://&lt;storage-account&gt;.queue.core.windows.net</code>
Table Storage	<code>https://&lt;storage-account&gt;.table.core.windows.net</code>

When your account is created with standard endpoints, you can easily construct the URL for an object in Azure Storage by appending the object's location in the storage account to the endpoint. For example, the URL for a blob will be similar to:

```
https://*mystorageaccount*.blob.core.windows.net/*mycontainer*/*myblob*
```

## Azure DNS zone endpoints (preview)

When you create an Azure Storage account with Azure DNS zone endpoints (preview), Azure Storage dynamically selects an Azure DNS zone and assigns it to the storage account when it is created. The new storage account's endpoints are created in the dynamically selected Azure DNS zone. For more information about Azure DNS zones, see [DNS zones](#).

An Azure DNS zone service endpoint in Azure Storage includes the protocol (HTTPS is recommended), the storage account name as the subdomain, and a domain that includes the name of the service and the identifier for the DNS zone. The identifier for the DNS zone always begins with `z` and can range from `z00` to `z99`.

The following table lists the format for Azure DNS Zone endpoints for each of the Azure Storage services, where the zone is `z5`.

STORAGE SERVICE	ENDPOINT
Blob Storage	<code>https://&lt;storage-account&gt;.z[00-99].blob.storage.azure.net</code>
Static website (Blob Storage)	<code>https://&lt;storage-account&gt;.z[00-99].web.storage.azure.net</code>
Data Lake Storage Gen2	<code>https://&lt;storage-account&gt;.z[00-99].dfs.storage.azure.net</code>
Azure Files	<code>https://&lt;storage-account&gt;.z[00-99].file.storage.azure.net</code>

STORAGE SERVICE	ENDPOINT
Queue Storage	<code>https://&lt;storage-account&gt;.z[00-99].queue.storage.azure.net</code>
Table Storage	<code>https://&lt;storage-account&gt;.z[00-99].table.storage.azure.net</code>

### IMPORTANT

You can create up to 5000 accounts with Azure DNS Zone endpoints per subscription. However, you may need to update your application code to query for the account endpoint at runtime. You can call the [Get Properties](#) operation to query for the storage account endpoints.

Azure DNS zone endpoints are supported for accounts created with the Azure Resource Manager deployment model only. For more information, see [Azure Resource Manager overview](#).

To learn how to create a storage account with Azure DNS Zone endpoints, see [Create a storage account](#).

#### About the preview

The Azure DNS zone endpoints preview is available in all public regions. The preview is not available in any government cloud regions.

To register for the preview, follow the instructions provided in [Set up preview features in Azure subscription](#).

Specify `PartitionedDnsPublicPreview` as the feature name and `Microsoft.Storage` as the provider namespace.

## Migrate a storage account

The following table summarizes and points to guidance on how to move, upgrade, or migrate a storage account:

MIGRATION SCENARIO	DETAILS
Move a storage account to a different subscription	Azure Resource Manager provides options for moving a resource to a different subscription. For more information, see <a href="#">Move resources to a new resource group or subscription</a> .
Move a storage account to a different resource group	Azure Resource Manager provides options for moving a resource to a different resource group. For more information, see <a href="#">Move resources to a new resource group or subscription</a> .
Move a storage account to a different region	To move a storage account, create a copy of your storage account in another region. Then, move your data to that account by using AzCopy, or another tool of your choice. For more information, see <a href="#">Move an Azure Storage account to another region</a> .
Upgrade to a general-purpose v2 storage account	You can upgrade a general-purpose v1 storage account or Blob Storage account to a general-purpose v2 account. Note that this action can't be undone. For more information, see <a href="#">Upgrade to a general-purpose v2 storage account</a> .

MIGRATION SCENARIO	DETAILS
Migrate a classic storage account to Azure Resource Manager	The Azure Resource Manager deployment model is superior to the classic deployment model in terms of functionality, scalability, and security. For more information about migrating a classic storage account to Azure Resource Manager, see the "Migration of storage accounts" section of <a href="#">Platform-supported migration of IaaS resources from classic to Azure Resource Manager</a> .

## Transfer data into a storage account

Microsoft provides services and utilities for importing your data from on-premises storage devices or third-party cloud storage providers. Which solution you use depends on the quantity of data you're transferring. For more information, see [Azure Storage migration overview](#).

## Storage account encryption

All data in your storage account is automatically encrypted on the service side. For more information about encryption and key management, see [Azure Storage encryption for data at rest](#).

## Storage account billing

Azure Storage bills based on your storage account usage. All objects in a storage account are billed together as a group. Storage costs are calculated according to the following factors:

- **Region** refers to the geographical region in which your account is based.
- **Account type** refers to the type of storage account you're using.
- **Access tier** refers to the data usage pattern you've specified for your general-purpose v2 or Blob Storage account.
- **Capacity** refers to how much of your storage account allotment you're using to store data.
- **Redundancy** determines how many copies of your data are maintained at one time, and in what locations.
- **Transactions** refer to all read and write operations to Azure Storage.
- **Data egress** refers to any data transferred out of an Azure region. When the data in your storage account is accessed by an application that isn't running in the same region, you're charged for data egress. For information about using resource groups to group your data and services in the same region to limit egress charges, see [What is an Azure resource group?](#).

The [Azure Storage pricing page](#) provides detailed pricing information based on account type, storage capacity, replication, and transactions. The [Data Transfers pricing details](#) provides detailed pricing information for data egress. You can use the [Azure Storage pricing calculator](#) to help estimate your costs.

Azure services cost money. Azure Cost Management helps you set budgets and configure alerts to keep spending under control. Analyze, manage, and optimize your Azure costs with Cost Management. To learn more, see the [quickstart on analyzing your costs](#).

## Legacy storage account types

The following table describes the legacy storage account types. These account types aren't recommended by Microsoft, but may be used in certain scenarios:

Type of legacy storage account	Supported storage services	Redundancy options	Deployment model	Usage
Standard general-purpose v1	Blob Storage, Queue Storage, Table Storage, and Azure Files	LRS/GRS/RA-GRS	Resource Manager, classic <sup>1</sup>	<p>General-purpose v1 accounts may not have the latest features or the lowest per-gigabyte pricing. Consider using it for these scenarios:</p> <ul style="list-style-type: none"> <li>• Your applications require the Azure <a href="#">classic deployment model</a><sup>1</sup>.</li> <li>• Your applications are transaction-intensive or use significant geo-replication bandwidth, but don't require large capacity. In this case, a general-purpose v1 account may be the most economical choice.</li> <li>• You use a version of the Azure Storage REST API that is earlier than February 14, 2014, or a client library with a version lower than 4.x, and you can't upgrade your application.</li> <li>• You're selecting a storage account to use as a cache for Azure Site Recovery. Because Site Recovery is transaction-intensive, a general-purpose v1 account may</li> </ul>

TYPE OF LEGACY STORAGE ACCOUNT	SUPPORTED STORAGE SERVICES	REDUNDANCY OPTIONS	DEPLOYMENT MODEL	be more cost-effective. For more information, see <a href="#">Support matrix for Azure VM disaster recovery between Azure regions</a> .
Standard Blob Storage	Blob Storage (block blobs and append blobs only)	LRS/GRS/RA-GRS	Resource Manager	Microsoft recommends using standard general-purpose v2 accounts instead when possible.

<sup>1</sup> Beginning August 1, 2022, you'll no longer be able to create new storage accounts with the classic deployment model. Resources created prior to that date will continue to be supported through August 31, 2024. For more information, see [Azure classic storage accounts will be retired on 31 August 2024](#).

## Scalability targets for standard storage accounts

The following table describes default limits for Azure general-purpose v2 (GPv2), general-purpose v1 (GPv1), and Blob storage accounts. The *ingress* limit refers to all data that is sent to a storage account. The *egress* limit refers to all data that is received from a storage account.

Microsoft recommends that you use a GPv2 storage account for most scenarios. You can easily upgrade a GPv1 or a Blob storage account to a GPv2 account with no downtime and without the need to copy data. For more information, see [Upgrade to a GPv2 storage account](#).

### NOTE

You can request higher capacity and ingress limits. To request an increase, contact [Azure Support](#).

RESOURCE	LIMIT
Maximum number of storage accounts with standard endpoints per region per subscription, including standard and premium storage accounts.	250
Maximum number of storage accounts with Azure DNS zone endpoints (preview) per region per subscription, including standard and premium storage accounts.	5000 (preview)
Default maximum storage account capacity	5 PiB <sup>1</sup>
Maximum number of blob containers, blobs, file shares, tables, queues, entities, or messages per storage account.	No limit
Default maximum request rate per storage account	20,000 requests per second <sup>1</sup>

RESOURCE	LIMIT
<p>Default maximum ingress per general-purpose v2 and Blob storage account in the following regions (LRS/GRS):</p> <ul style="list-style-type: none"> <li>• Australia East</li> <li>• Central US</li> <li>• East Asia</li> <li>• East US 2</li> <li>• Japan East</li> <li>• Korea Central</li> <li>• North Europe</li> <li>• South Central US</li> <li>• Southeast Asia</li> <li>• UK South</li> <li>• West Europe</li> <li>• West US</li> </ul>	60 Gbps <sup>1</sup>
<p>Default maximum ingress per general-purpose v2 and Blob storage account in the following regions (ZRS):</p> <ul style="list-style-type: none"> <li>• Australia East</li> <li>• Central US</li> <li>• East US</li> <li>• East US 2</li> <li>• Japan East</li> <li>• North Europe</li> <li>• South Central US</li> <li>• Southeast Asia</li> <li>• UK South</li> <li>• West Europe</li> <li>• West US 2</li> </ul>	60 Gb ps <sup>1</sup>
Default maximum ingress per general-purpose v2 and Blob storage account in regions that aren't listed in the previous row.	25 Gbps <sup>1</sup>
Default maximum ingress for general-purpose v1 storage accounts (all regions)	10 Gbps <sup>1</sup>
<p>Default maximum egress for general-purpose v2 and Blob storage accounts in the following regions (LRS/GRS):</p> <ul style="list-style-type: none"> <li>• Australia East</li> <li>• Central US</li> <li>• East Asia</li> <li>• East US 2</li> <li>• Japan East</li> <li>• Korea Central</li> <li>• North Europe</li> <li>• South Central US</li> <li>• Southeast Asia</li> <li>• UK South</li> <li>• West Europe</li> <li>• West US</li> </ul>	120 Gbps <sup>1</sup>

RESOURCE	LIMIT
Default maximum egress for general-purpose v2 and Blob storage accounts in the following regions (ZRS): <ul style="list-style-type: none"> <li>• Australia East</li> <li>• Central US</li> <li>• East US</li> <li>• East US 2</li> <li>• Japan East</li> <li>• North Europe</li> <li>• South Central US</li> <li>• Southeast Asia</li> <li>• UK South</li> <li>• West Europe</li> <li>• West US 2</li> </ul>	120 Gbps <sup>1</sup>
Default maximum egress for general-purpose v2 and Blob storage accounts in regions that aren't listed in the previous row.	50 Gbps <sup>1</sup>
Maximum number of IP address rules per storage account	200
Maximum number of virtual network rules per storage account	200
Maximum number of resource instance rules per storage account	200
Maximum number of private endpoints per storage account	200

<sup>1</sup> Azure Storage standard accounts support higher capacity limits and higher limits for ingress and egress by request. To request an increase in account limits, contact [Azure Support](#).

## Next steps

- [Create a storage account](#)
- [Upgrade to a general-purpose v2 storage account](#)
- [Recover a deleted storage account](#)

# Premium block blob storage accounts

8/22/2022 • 13 minutes to read • [Edit Online](#)

Premium block blob storage accounts make data available via high-performance hardware. Data is stored on solid-state drives (SSDs) which are optimized for low latency. SSDs provide higher throughput compared to traditional hard drives. File transfer is much faster because data is stored on instantly accessible memory chips. All parts of a drive accessible at once. By contrast, the performance of a hard disk drive (HDD) depends on the proximity of data to the read/write heads.

## High performance workloads

Premium block blob storage accounts are ideal for workloads that require fast and consistent response times and/or have a high number of input output operations per second (IOP). Example workloads include:

- **Interactive workloads.** Highly interactive and real-time applications must write data quickly. E-commerce and mapping applications often require instant updates and user feedback. For example, in an e-commerce application, less frequently viewed items are likely not cached. However, they must be instantly displayed to the customer on demand. Interactive editing or multi-player online gaming applications maintain a quality experience by providing real-time updates.
- **IoT/ streaming analytics.** In an IoT scenario, lots of smaller write operations might be pushed to the cloud every second. Large amounts of data might be taken in, aggregated for analysis purposes, and then deleted almost immediately. The high ingestion capabilities of premium block blob storage make it efficient for this type of workload.
- **Artificial intelligence/machine learning (AI/ML).** AI/ML deals with the consumption and processing of different data types like visuals, speech, and text. This high-performance computing type of workload deals with large amounts of data that requires rapid response and efficient ingestion times for data analysis.

## Cost effectiveness

Premium block blob storage accounts have a higher storage cost but a lower transaction cost as compared to standard general-purpose v2 accounts. If your applications and workloads execute a large number of transactions, premium blob blob storage can be cost-effective, especially if the workload is write-heavy.

In most cases, workloads executing more than 35 to 40 transactions per second per terabyte (TPS/TB) are good candidates for this type of account. For example, if your workload executes 500 million read operations and 100 million write operations in a month, then you can calculate the TPS/TB as follows:

- Write transactions per second =  $100,000,000 / (30 \times 24 \times 60 \times 60) = 39$  (*rounded to the nearest whole number*)
- Read transactions per second =  $500,000,000 / (30 \times 24 \times 60 \times 60) = 193$  (*rounded to the nearest whole number*)
- Total transactions per second =  $193 + 39 = 232$
- Assuming your account had 5TB data on average, then TPS/TB would be  $230 / 5 = 46$ .

**NOTE**

Prices differ per operation and per region. Use the [Azure pricing calculator](#) to compare pricing between standard and premium performance tiers.

The following table demonstrates the cost-effectiveness of premium block blob storage accounts. The numbers in this table are based on a Azure Data Lake Storage Gen2 enabled premium block blob storage account (also referred to as the [premium tier for Azure Data Lake Storage](#)). Each column represents the number of transactions in a month. Each row represents the percentage of transactions that are read transactions. Each cell in the table shows the percentage of cost reduction associated with a read transaction percentage and the number of transactions executed.

For example, assuming that your account is in the East US 2 region, the number of transactions with your account exceeds 90M, and 70% of those transactions are read transactions, premium block blob storage accounts are more cost-effective.

	30k	100k	500k	10M	50M	70M	80M	90M	100M	200M	500M	1B
0%	-723%	-704%	-612%	-111%	22%	34%	38%	41%	43%	54%	60%	63%
10%	-723%	-706%	-621%	-125%	18%	31%	35%	38%	41%	53%	60%	62%
20%	-724%	-709%	-631%	-141%	13%	27%	32%	35%	38%	51%	59%	62%
30%	-725%	-711%	-641%	-160%	7%	23%	28%	32%	35%	50%	59%	62%
40%	-726%	-714%	-651%	-182%	-1%	17%	23%	27%	31%	47%	58%	61%
50%	-727%	-716%	-662%	-210%	-11%	9%	16%	21%	25%	45%	57%	61%
60%	-727%	-719%	-673%	-244%	-25%	-1%	6%	12%	17%	40%	55%	60%
70%	-728%	-721%	-684%	-289%	-45%	-17%	-7%	0%	6%	34%	52%	59%
80%	-729%	-724%	-696%	-348%	-76%	-41%	-30%	-20%	-12%	24%	48%	56%
90%	-730%	-726%	-708%	-431%	-133%	-87%	-71%	-59%	-48%	4%	39%	52%
100%	-730%	-729%	-720%	-556%	-266%	-203%	-180%	-160%	-144%	-55%	12%	38%
TPS/TB	0.01	0.04	0.19	3.86	19.29	27.01	30.86	34.72	38.58	77.2	192.9	385.8

**NOTE**

If you prefer to evaluate cost effectiveness based on the number of transactions per second for each TB of data, you can use the column headings that appear at the bottom of the table.

## Premium scenarios

This section contains real-world examples of how some of our Azure Storage partners use premium block blob storage. Some of them also enable Azure Data Lake Storage Gen2 which introduces a hierarchical file structure that can further enhance transaction performance in certain scenarios.

**TIP**

If you have an analytics use case, we highly recommend that you use Azure Data Lake Storage Gen2 along with a premium block blob storage account.

This section contains the following examples:

- [Premium block blob storage accounts](#)
  - [High performance workloads](#)
  - [Cost effectiveness](#)
  - [Premium scenarios](#)
  - [Fast data hydration](#)

- [Interactive editing applications](#)
- [Data visualization software](#)
- [E-commerce businesses](#)
- [Interactive analytics](#)
- [Data processing pipelines](#)
- [Internet of Things \(IoT\)](#)
- [Machine Learning](#)
- [Real-time streaming analytics](#)
- [Getting started with premium](#)
  - [Check for Blob Storage feature compatibility](#)
  - [Create a new Storage account](#)
- [See also](#)

## **Fast data hydration**

Premium block blob storage can help you *hydrate* or bring up your environment quickly. In industries such as banking, certain regulatory requirements might require companies to regularly tear down their environments, and then bring them back up from scratch. The data used to hydrate their environment must load quickly.

Some of our partners store a copy of their MongoDB instance each week to a premium block blob storage account. The system is then torn down. To get the system back online quickly again, the latest copy of the MongoDB instance is read and loaded. For audit purposes, previous copies are maintained in cloud storage for a period of time.

## **Interactive editing applications**

In applications where multiple users edit the same content, the speed of updates becomes critical for a smooth user experience.

Some of our partners develop video editing software. Any update that a user makes to a video is immediately visible to other users. Users can focus on their tasks instead of waiting for content updates to appear. The low latencies associated with premium block blob storage helps to create this seamless and collaborative experience.

## **Data visualization software**

Users can be far more productive with data visualization software if rendering time is quick.

We've seen companies in the mapping industry use mapping editors to detect issues with maps. These editors use data that is generated from customer Global Positioning System (GPS) data. To create map overlaps, the editing software renders small sections of a map by quickly performing key lookups.

In one case, before using premium block blob storage, a partner used HBase clusters backed by standard general-purpose v2 storage. However, it became expensive to keep large clusters running all of the time. This partner decided to move away from this architecture, and instead used premium block blob storage for fast key lookups. To create overlaps, they used REST APIs to render tiles corresponding to GPS coordinates. The premium block blob storage account provided them with a cost-effective solution, and latencies were far more predictable.

## **E-commerce businesses**

In addition to supporting their customer facing stores, e-commerce businesses might also provide data warehousing and analytics solutions to internal teams. We've seen partners use premium block blob storage accounts to support the low latency requirements by these data warehousing and analytics solutions. In one case, a catalog team maintains a data warehousing application for data that pertains to offers, pricing, ship methods, suppliers, inventory, and logistics. Information is queried, scanned, extracted, and mined for multiple use cases. The team runs analytics on this data to provide various merchandising teams with relevant insights and information.

## Interactive analytics

In almost every industry, there is a need for enterprises to query and analyze their data interactively.

Data scientists, analysts, and developers can derive time-sensitive insights faster by running queries on data that is stored in a premium block blob storage account. Executives can load their dashboards much more quickly when the data that appears in those dashboards comes from a premium block blob storage account instead of a standard general-purpose v2 account.

In one scenario, analysts needed to analyze telemetry data from millions of devices quickly to better understand how their products are used, and to make product release decisions. Storing data in SQL databases is expensive. To reduce cost, and to increase queryable surface area, they used an Azure Data Lake Storage Gen2 enabled premium block blob storage account and performed computation in Presto and Spark to produce insights from hive tables. This way, even rarely accessed data has all of the same power of compute as frequently accessed data.

To close the gap between SQL's subsecond performance and Presto's input output operations per second (IOPs) to external storage, consistency and speed are critical, especially when dealing with small optimized row columnar (ORC) files. A premium block blob storage account when used with Data Lake Storage Gen2, has repeatedly demonstrated a 3X performance improvement over a standard general-purpose v2 account in this scenario. Queries executed fast enough to feel local to the compute machine.

In another case, a partner stores and queries logs that are generated from their security solution. The logs are generated by using Databricks, and then stored in a Data Lake Storage Gen2 enabled premium block blob storage account. End users query and search this data by using Azure Data Explorer. They chose this type of account to increase stability and increase the performance of interactive queries. They also set the life cycle management `Delete Action` policy to a few days, which helps to reduce costs. This policy prevents them from keeping the data forever. Instead, data is deleted once it is no longer needed.

## Data processing pipelines

In almost every industry, there is a need for enterprises to process data. Raw data from multiple sources needs to be cleansed and processed so that it becomes useful for downstream consumption in tools such as data dashboards that help users make decisions.

While speed of processing is not always the top concern when processing data, some industries require it. For example, companies in the financial services industry often need to process data reliably and in the quickest way possible. To detect fraud, those companies must process inputs from various sources, identify risks to their customers, and take swift action.

In some cases, we've seen partners use multiple standard storage accounts to store data from various sources. Some of this data is then moved to a Data Lake Storage enabled premium block blob storage account where a data processing application frequently reads newly arriving data. Directory listing calls in this account were much faster and performed much more consistently than they would otherwise perform in a standard general-purpose v2 account. The speed and consistency offered by the account ensured that new data was always made available to downstream processing systems as quickly as possible. This helped them catch and act upon potential security risks promptly.

## Internet of Things (IoT)

IoT has become a significant part of our daily lives. IoT is used to track car movements, control lights, and monitor our health. It also has industrial applications. For example, companies use IoT to enable their smart factory projects, improve agricultural output, and on oil rigs for predictive maintenance. Premium block blob storage accounts add significant value to these scenarios.

We have partners in the mining industry. They use a Data Lake Storage Gen2 enable premium block blob storage account along with HDInsight (Hbase) to ingest time series sensor data from multiple mining equipment types, with a very taxing load profile. Premium block blob storage has helped to satisfy their need for high

sample rate ingestion. It's also cost effective, because premium block blob storage is cost optimized for workloads that perform a large number of write transactions, and this workload generates a large number of small write transactions (in the tens of thousands per second).

## Machine Learning

In many cases, a lot of data has to be processed to train a machine learning model. To complete this processing, compute machines must run for a long time. Compared to storage costs, compute costs usually account for a much larger percentage of your bill, so reducing the amount of time that your compute machines run can lead to significant savings. The low latency that you get by using premium block blob storage can significantly reduce this time and your bill.

We have partners that deploy data processing pipelines to spark clusters where they run machine learning training and inference. They store spark tables (parquet files) and checkpoints to a premium block blob storage account. Spark checkpoints can create a huge number of nested files and folders. Their directory listing operations are fast because they combined the low latency of a premium block blob storage account with the hierarchical data structure made available with Data Lake Storage Gen2.

We also have partners in the semiconductor industry with use cases that intersect IoT and machine learning. IoT devices attached to machines in the manufacturing plant take images of semiconductor wafers and send those to their account. Using deep learning inference, the system can inform the on-premises machines if there is an issue with the production and if an action needs to be taken. They must be able to load and process images quickly and reliably. Using Data Lake Storage Gen2 enabled premium block blob storage account helps to make this possible.

## Real-time streaming analytics

To support interactive analytics in near real time, a system must ingest and process large amounts of data, and then make that data available to downstream systems. Using a Data Lake Storage Gen2 enabled premium block blob storage account is perfect for these types of scenarios.

Companies in the media and entertainment industry can generate a large number of logs and telemetry data in a short amount of time as they broadcast an event. Some of our partners rely on multiple content delivery network (CDN) partners for streaming. They must make near real-time decisions about which CDN partners to allocate traffic to. Therefore, data needs to be available for querying a few seconds after it is ingested. To facilitate this quick decision making, they use data stored within premium block blob storage, and process that data in Azure Data Explorer (ADX). All of the telemetry that is uploaded to storage is transformed in ADX, where it can be stored in a familiar format that operators and executives can query quickly and reliably.

Data is uploaded into multiple premium performance Blob Storage accounts. Each account is connected to an Event Grid and Event Hub resource. ADX retrieves the data from Blob Storage, performs any required transformations to normalize the data (For example: decompressing zip files or converting from JSON to CSV). Then, the data is made available for query through ADX and dashboards displayed in Grafana. Grafana dashboards are used by operators, executives, and other users. The customer retains their original logs in premium performance storage, or they copy them to a general-purpose v2 storage account where they can be stored in the hot or cool access tier for long-term retention and future analysis.

## Getting started with premium

First, check to make sure your favorite Blob Storage features are compatible with premium block blob storage accounts, then create the account.

#### NOTE

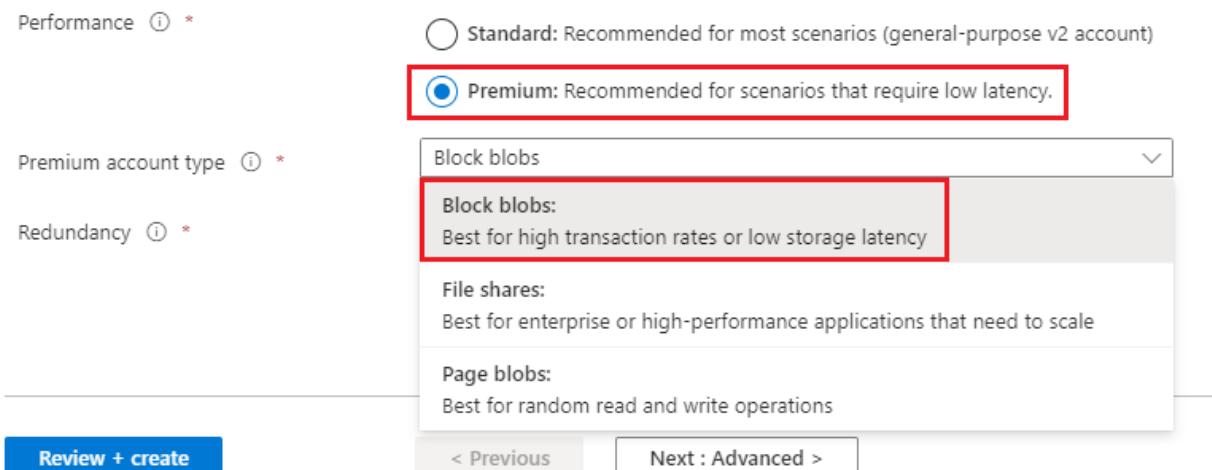
You can't convert an existing standard general-purpose v2 storage account to a premium block blob storage account. To migrate to a premium block blob storage account, you must create a premium block blob storage account, and migrate the data to the new account.

### Check for Blob Storage feature compatibility

Some Blob Storage features aren't yet supported or have partial support in premium block blob storage accounts. Before choosing premium, review the [Blob Storage feature support in Azure Storage accounts](#) article to determine whether the features that you intend to use are fully supported in your account. Feature support is always expanding so make sure to periodically review this article for updates.

### Create a new Storage account

To create a premium block blob storage account, make sure to choose the **Premium** performance option and the **Block blobs** account type as you create the account.



#### NOTE

Some Blob Storage features aren't yet supported or have partial support in premium block blob storage accounts. Before choosing premium, review the [Blob Storage feature support in Azure Storage accounts](#) article to determine whether the features that you intend to use are fully supported in your account. Feature support is always expanding so make sure to periodically review this article for updates.

If your storage account is going to be used for analytics, we highly recommend that you use Azure Data Lake Storage Gen2 along with a premium block blob storage account. To unlock Azure Data Lake Storage Gen2 capabilities, enable the **Hierarchical namespace** setting in the **Advanced** tab of the **Create storage account** page.

The following image shows this setting in the **Create storage account** page.

[Home](#) > [Create a resource](#) >

## Create a storage account

Basics

Advanced 

Networking

Data protection

Tags

Review + create

 Certain options have been disabled by default due to the combination of storage account performance, redundancy, and region.

### Security

Configure security settings that impact your storage account.

Enable secure transfer 



Enable infrastructure encryption 



 Sign up is currently required to enable infrastructure encryption on a per-subscription basis. [Sign up](#)

Enable blob public access 



Enable storage account key access 



Minimum TLS version 

Version 1.2 

### Data Lake Storage Gen2

The Data Lake Storage Gen2 hierarchical namespace accelerates big data analytics workloads and enables file-level access control lists (ACLs). [Learn more](#)

Enable hierarchical namespace 



For complete guidance, see [Create a storage account](#) account.

## See also

- [Storage account overview](#)
- [Introduction to Azure Data Lake Storage Gen2](#)
- [Create a storage account to use with Azure Data Lake Storage Gen2](#)
- [Premium tier for Azure Data Lake Storage](#)

# Authorize access to data in Azure Storage

8/22/2022 • 4 minutes to read • [Edit Online](#)

Each time you access data in your storage account, your client application makes a request over HTTP/HTTPS to Azure Storage. By default, every resource in Azure Storage is secured, and every request to a secure resource must be authorized. Authorization ensures that the client application has the appropriate permissions to access a particular resource in your storage account.

## Understand authorization for data operations

The following table describes the options that Azure Storage offers for authorizing access to data:

AZURE ARTIFACT	SHARED KEY (STORAGE ACCOUNT KEY)	SHARED ACCESS SIGNATURE (SAS)	AZURE ACTIVE DIRECTORY (AZURE AD)	ON-PREMISES ACTIVE DIRECTORY DOMAIN SERVICES	ANONYMOUS PUBLIC READ ACCESS	STORAGE LOCAL USERS
Azure Blobs	Supported	Supported	Supported	Not supported	Supported	Supported, only for SFTP
Azure Files (SMB)	Supported	Not supported	Supported, only with AAD Domain Services	Supported, credentials must be synced to Azure AD	Not supported	Supported
Azure Files (REST)	Supported	Supported	Not supported	Not supported	Not supported	Not supported
Azure Queues	Supported	Supported	Supported	Not Supported	Not supported	Not supported
Azure Tables	Supported	Supported	Supported	Not supported	Not supported	Not supported

Each authorization option is briefly described below:

- **Shared Key authorization** for blobs, files, queues, and tables. A client using Shared Key passes a header with every request that is signed using the storage account access key. For more information, see [Authorize with Shared Key](#).

Microsoft recommends that you disallow Shared Key authorization for your storage account. When Shared Key authorization is disallowed, clients must use Azure AD or a user delegation SAS to authorize requests for data in that storage account. For more information, see [Prevent Shared Key authorization for an Azure Storage account](#).

- **Shared access signatures** for blobs, files, queues, and tables. Shared access signatures (SAS) provide limited delegated access to resources in a storage account via a signed URL. The signed URL specifies the permissions granted to the resource and the interval over which the signature is valid. A service SAS or account SAS is signed with the account key, while the user delegation SAS is signed with Azure AD credentials and applies to blobs only. For more information, see [Using shared access signatures \(SAS\)](#).
- **Azure Active Directory (Azure AD) integration** for authorizing requests to blob, queue, and table

resources. Microsoft recommends using Azure AD credentials to authorize requests to data when possible for optimal security and ease of use. For more information about Azure AD integration, see the articles for either [blob](#), [queue](#), or [table](#) resources.

You can use Azure role-based access control (Azure RBAC) to manage a security principal's permissions to blob, queue, and table resources in a storage account. You can additionally use Azure attribute-based access control (ABAC) to add conditions to Azure role assignments for blob resources. For more information about RBAC, see [What is Azure role-based access control \(Azure RBAC\)?](#). For more information about ABAC, see [What is Azure attribute-based access control \(Azure ABAC\)? \(preview\)](#).

- **Azure Active Directory Domain Services (Azure AD DS) authentication** for Azure Files. Azure Files supports identity-based authorization over Server Message Block (SMB) through Azure AD DS. You can use Azure RBAC for fine-grained control over a client's access to Azure Files resources in a storage account. For more information about Azure Files authentication using domain services, see the [overview](#).
- **On-premises Active Directory Domain Services (AD DS, or on-premises AD DS) authentication** for Azure Files. Azure Files supports identity-based authorization over SMB through AD DS. Your AD DS environment can be hosted in on-premises machines or in Azure VMs. SMB access to Files is supported using AD DS credentials from domain joined machines, either on-premises or in Azure. You can use a combination of Azure RBAC for share level access control and NTFS DACLs for directory/file level permission enforcement. For more information about Azure Files authentication using domain services, see the [overview](#).
- **Anonymous public read access** for containers and blobs. When anonymous access is configured, then clients can read blob data without authorization. For more information, see [Manage anonymous read access to containers and blobs](#).

You can disallow anonymous public read access for a storage account. When anonymous public read access is disallowed, then users cannot configure containers to enable anonymous access, and all requests must be authorized. For more information, see [Prevent anonymous public read access to containers and blobs](#).

- **Storage Local Users** can be used to access blobs with SFTP or files with SMB. Storage Local Users support container level permissions for authorization. See [Connect to Azure Blob Storage by using the SSH File Transfer Protocol \(SFTP\)](#) for more information on how Storage Local Users can be used with SFTP.

## Protect your access keys

Your storage account access keys are similar to a root password for your storage account. Always be careful to protect your access keys. Use Azure Key Vault to manage and rotate your keys securely. Avoid distributing access keys to other users, hard-coding them, or saving them anywhere in plain text that is accessible to others. Rotate your keys if you believe they may have been compromised.

### NOTE

Microsoft recommends using Azure Active Directory (Azure AD) to authorize requests against blob and queue data if possible, rather than using the account keys (Shared Key authorization). Authorization with Azure AD provides superior security and ease of use over Shared Key authorization.

To protect an Azure Storage account with Azure AD Conditional Access policies, you must disallow Shared Key authorization for the storage account. For more information about how to disallow Shared Key authorization, see [Prevent Shared Key authorization for an Azure Storage account](#).

## Next steps

- Authorize access with Azure Active Directory to either [blob](#), [queue](#), or [table](#) resources.
- [Authorize with Shared Key](#)
- [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#)

# Authorize access to blobs using Azure Active Directory

8/22/2022 • 8 minutes to read • [Edit Online](#)

Azure Storage supports using Azure Active Directory (Azure AD) to authorize requests to blob data. With Azure AD, 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 Azure AD to return an OAuth 2.0 token. The token can then be used to authorize a request against the Blob service.

Authorizing requests against Azure Storage with Azure AD provides superior security and ease of use over Shared Key authorization. Microsoft recommends using Azure AD authorization with your blob applications when possible to assure access with minimum required privileges.

Authorization with Azure AD is available for all general-purpose and Blob storage accounts in all public regions and national clouds. Only storage accounts created with the Azure Resource Manager deployment model support Azure AD authorization.

Blob storage additionally supports creating shared access signatures (SAS) that are signed with Azure AD credentials. For more information, see [Grant limited access to data with shared access signatures](#).

## Overview of Azure AD for blobs

When a security principal (a user, group, or application) attempts to access a blob resource, the request must be authorized, unless it is a blob available for anonymous access. With Azure AD, access to a resource is a two-step process. First, the security principal's identity is authenticated and an OAuth 2.0 token is returned. Next, the token is passed as part of a request to the Blob service and used by the service to authorize access to the specified resource.

The authentication step requires that an application request an OAuth 2.0 access token at runtime. If an application is running from within an Azure entity such as an Azure VM, a virtual machine scale set, or an Azure Functions app, it can use a [managed identity](#) to access blob data. To learn how to authorize requests made by a managed identity to the Azure Blob service, see [Authorize access to blob data with managed identities for Azure resources](#).

The authorization step requires that one or more Azure RBAC roles be assigned to the security principal making the request. For more information, see [Assign Azure roles for access rights](#).

Native applications and web applications that make requests to the Azure Blob service can also authorize access with Azure AD. To learn how to request an access token and use it to authorize requests for blob data, see [Authorize access to Azure Storage with Azure AD from an Azure Storage application](#).

Authorizing blob data operations with Azure AD is supported only for REST API versions 2017-11-09 and later. For more information, see [Versioning for the Azure Storage services](#).

## Assign Azure roles for access rights

Azure Active Directory (Azure AD) authorizes access rights to secured resources through Azure RBAC. Azure Storage defines a set of built-in RBAC roles that encompass common sets of permissions used to access blob data. You can also define custom roles for access to blob data. To learn more about assigning Azure roles for blob access, see [Assign an Azure role for access to blob data](#).

An Azure AD security principal may be a user, a group, an application service principal, or a [managed identity for Azure resources](#). The RBAC roles that are assigned to a security principal determine the permissions that the principal will have. To learn more about assigning Azure roles for blob access, see [Assign an Azure role for access to blob data](#)

In some cases you may need to enable fine-grained access to blob resources or to simplify permissions when you have a large number of role assignments for a storage resource. You can use Azure attribute-based access control (Azure ABAC) to configure conditions on role assignments. You can use conditions with a [custom role](#) or select built-in roles. For more information about configuring conditions for Azure storage resources with ABAC, see [Authorize access to blobs using Azure role assignment conditions \(preview\)](#). For details about supported conditions for blob data operations, see [Actions and attributes for Azure role assignment conditions in Azure Storage \(preview\)](#).

## Resource scope

Before you assign an Azure RBAC role to a security principal, determine the scope of access that the security principal should have. Best practices dictate that it's always best to grant only the narrowest possible scope. Azure RBAC roles defined at a broader scope are inherited by the resources beneath them.

You can scope access to Azure blob resources at the following levels, beginning with the narrowest scope:

- **An individual container.** At this scope, a role assignment applies to all of the blobs in the container, as well as container properties and metadata.
- **The storage account.** At this scope, a role assignment applies to all containers and their blobs.
- **The resource group.** At this scope, a role assignment applies to all of the containers in all of the storage accounts in the resource group.
- **The subscription.** At this scope, a role assignment applies to all of the containers in all of the storage accounts in all of the resource groups in the subscription.
- **A management group.** At this scope, a role assignment applies to all of the containers in all of the storage accounts in all of the resource groups in all of the subscriptions in the management group.

For more information about scope for Azure RBAC role assignments, see [Understand scope for Azure RBAC](#).

## Azure built-in roles for blobs

Azure RBAC provides a number of built-in roles for authorizing access to blob data using Azure AD and OAuth. Some examples of roles that provide permissions to data resources in Azure Storage include:

- **Storage Blob Data Owner:** Use to set ownership and manage POSIX access control for Azure Data Lake Storage Gen2. For more information, see [Access control in Azure Data Lake Storage Gen2](#).
- **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 Azure AD credentials for a container or blob.

To learn how to assign an Azure built-in role to a security principal, see [Assign an Azure role for access to blob data](#). To learn how to list Azure RBAC roles and their permissions, see [List Azure role definitions](#).

For more information about how built-in roles are defined for Azure Storage, see [Understand role definitions](#). For information about creating Azure custom roles, see [Azure custom roles](#).

Only roles explicitly defined for data access permit a security principal to access blob data. Built-in roles such as **Owner**, **Contributor**, and **Storage Account Contributor** permit a security principal to manage a storage account, but do not provide access to the blob data within that account via Azure AD. However, if a role includes **Microsoft.Storage/storageAccounts/listKeys/action**, then a user to whom that role is assigned can access data in the storage account via Shared Key authorization with the account access keys. For more information, see [Choose how to authorize access to blob data in the Azure portal](#).

For detailed information about Azure built-in roles for Azure Storage for both the data services and the management service, see the **Storage** section in [Azure built-in roles for Azure RBAC](#). Additionally, for information about the different types of roles that provide permissions in Azure, see [Classic subscription administrator roles](#), [Azure roles](#), and [Azure AD roles](#).

**IMPORTANT**

Azure role assignments may take up to 30 minutes to propagate.

## Access permissions for data operations

For details on the permissions required to call specific Blob service operations, see [Permissions for calling data operations](#).

# Access data with an Azure AD account

Access to blob data via the Azure portal, PowerShell, or Azure CLI can be authorized either by using the user's Azure AD account or by using the account access keys (Shared Key authorization).

### Data access from the Azure portal

The Azure portal can use either your Azure AD account or the account access keys to access blob data in an Azure storage account. Which authorization scheme the Azure portal uses depends on the Azure roles that are assigned to you.

When you attempt to access blob data, the Azure portal first checks whether you have been assigned an Azure role with **Microsoft.Storage/storageAccounts/listkeys/action**. If you have been assigned a role with this action, then the Azure portal uses the account key for accessing blob data via Shared Key authorization. If you have not been assigned a role with this action, then the Azure portal attempts to access data using your Azure AD account.

To access blob data from the Azure portal using your Azure AD account, you need permissions to access blob data, and you also need permissions to navigate through the storage account resources in the Azure portal. The built-in roles provided by Azure Storage grant access to blob resources, but they don't grant permissions to storage account resources. For this reason, access to the portal also requires the assignment of an Azure Resource Manager role such as the **Reader** role, scoped to the level of the storage account or higher. The **Reader** role grants the most restricted permissions, but another Azure Resource Manager role that grants access to storage account management resources is also acceptable. To learn more about how to assign permissions to users for data access in the Azure portal with an Azure AD account, see [Assign an Azure role for access to blob data](#).

The Azure portal indicates which authorization scheme is in use when you navigate to a container. For more information about data access in the portal, see [Choose how to authorize access to blob data in the Azure portal](#).

### Data access from PowerShell or Azure CLI

Azure CLI and PowerShell support signing in with Azure AD credentials. After you sign in, your session runs under those credentials. To learn more, see one of the following articles:

- [Choose how to authorize access to blob data with Azure CLI](#)
- [Run PowerShell commands with Azure AD credentials to access blob data](#)

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess

support for this feature.

## Next steps

- [Authorize access to data in Azure Storage](#)
- [Assign an Azure role for access to blob data](#)

# Authorize access to blobs using Azure role assignment conditions (preview)

8/22/2022 • 4 minutes to read • [Edit Online](#)

## IMPORTANT

Azure ABAC and Azure role assignment conditions are currently in preview. This preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Attribute-based access control (ABAC) is an authorization strategy that defines access levels based on attributes associated with security principals, resources, requests, and the environment. With ABAC, you can grant a security principal access to a resource based on a condition expressed as a predicate using these attributes.

Azure ABAC builds on Azure role-based access control (Azure RBAC) by adding [conditions to Azure role assignments](#). This preview includes support for role assignment conditions on Blobs and Data Lake Storage Gen2. It enables you to author role-assignment conditions based on principal, resource and request attributes.

## Overview of conditions in Azure Storage

Azure Storage enables the [use of Azure Active Directory](#) (Azure AD) to authorize requests to blob, queue, and table resources using Azure RBAC. Azure RBAC helps you manage access to resources by defining who has access to resources and what they can do with those resources, using role definitions and role assignments. Azure Storage defines a set of Azure [built-in roles](#) that encompass common sets of permissions used to access blob, queue and table data. You can also define custom roles with select set of permissions. Azure Storage supports role assignments for storage accounts or blob containers.

Azure ABAC builds on Azure RBAC by adding role assignment conditions in the context of specific actions. A *role assignment condition* is an additional check that is evaluated when the action on the storage resource is being authorized. This condition is expressed as a predicate using attributes associated with any of the following:

- Security principal that is requesting authorization
- Resource to which access is being requested
- Parameters of the request
- Environment from which the request originates

The benefits of using role assignment conditions are:

- **Enable finer-grained access to resources** - For example, if you want to grant a user read access to blobs in your storage accounts only if the blobs are tagged as Project=Sierra, you can use conditions on the read action using tags as an attribute.
- **Reduce the number of role assignments you have to create and manage** - You can do this by using a generalized role assignment for a security group, and then restricting the access for individual members of the group using a condition that matches attributes of a principal with attributes of a specific resource being accessed (such as, a blob or a container).
- **Express access control rules in terms of attributes with business meaning** - For example, you can express your conditions using attributes that represent a project name, business application, organization function, or classification level.

The tradeoff of using conditions is that you need a structured and consistent taxonomy when using attributes across your organization. Attributes must be protected to prevent access from being compromised. Also, conditions must be carefully designed and reviewed for their effect.

Role-assignment conditions in Azure Storage are supported for blobs. You can use conditions with accounts that have the [hierarchical namespace](#) (HNS) feature enabled on them. Conditions are currently not supported for queue, table, or file resources in Azure Storage.

## Supported attributes and operations

You can configure conditions on role assignments for [DataActions](#) to achieve these goals. You can use conditions with a [custom role](#) or select built-in roles. Note, conditions are not supported for management [Actions](#) through the [Storage resource provider](#).

In this preview, you can add conditions to built-in roles or custom roles. The built-in roles on which you can use role-assignment conditions in this preview include:

- [Storage Blob Data Reader](#)
- [Storage Blob Data Contributor](#)
- [Storage Blob Data Owner](#).
- [Storage Queue Data Contributor](#)
- [Storage Queue Data Message Processor](#)
- [Storage Queue Data Message Sender](#)
- [Storage Queue Data Reader](#)

You can use conditions with custom roles so long as the role includes [actions that support conditions](#).

If you're working with conditions based on [blob index tags](#), you should use the *Storage Blob Data Owner* since permissions for tag operations are included in this role.

### NOTE

Blob index tags are not supported for Data Lake Storage Gen2 storage accounts, which use a hierarchical namespace. You should not author role-assignment conditions using index tags on storage accounts that have HNS enabled.

The [Azure role assignment condition format](#) allows use of `@Principal`, `@Resource` or `@Request` attributes in the conditions. A `@Principal` attribute is a custom security attribute on a principal, such as a user, enterprise application (service principal), or managed identity. A `@Resource` attribute refers to an existing attribute of a storage resource that is being accessed, such as a storage account, a container, or a blob. A `@Request` attribute refers to an attribute or parameter included in a storage operation request.

Azure RBAC currently supports 2,000 role assignments in a subscription. If you need to create thousands of Azure role assignments, you may encounter this limit. Managing hundreds or thousands of role assignments can be difficult. In some cases, you can use conditions to reduce the number of role assignments on your storage account and make them easier to manage. You can [scale the management of role assignments](#) using conditions and [Azure AD custom security attributes](#) for principals.

## Next steps

- [Prerequisites for Azure role assignment conditions](#)
- [Tutorial: Add a role assignment condition to restrict access to blobs using the Azure portal](#)
- [Actions and attributes for Azure role assignment conditions in Azure Storage \(preview\)](#)
- [Example Azure role assignment conditions \(preview\)](#)

- [Troubleshoot Azure role assignment conditions \(preview\)](#)

## See also

- [What is Azure attribute-based access control \(Azure ABAC\)? \(preview\)](#)
- [FAQ for Azure role assignment conditions \(preview\)](#)
- [Azure role assignment condition format and syntax \(preview\)](#)
- [Scale the management of Azure role assignments by using conditions and custom security attributes \(preview\)](#)
- [Security considerations for Azure role assignment conditions in Azure Storage \(preview\)](#)

# Actions and attributes for Azure role assignment conditions in Azure Storage (preview)

8/22/2022 • 13 minutes to read • [Edit Online](#)

## IMPORTANT

Azure ABAC and Azure role assignment conditions are currently in preview. This preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

This article describes the supported attribute dictionaries that can be used in conditions on Azure role assignments for each Azure Storage [DataAction](#). For the list of Blob service operations that are affected by a specific permission or DataAction, see [Permissions for Blob service operations](#).

To understand the role assignment condition format, see [Azure role assignment condition format and syntax](#).

## Suboperations

Multiple Storage service operations can be associated with a single permission or DataAction. However, each of these operations that are associated with the same permission might support different parameters.

*Suboperations* enable you to differentiate between service operations that require the same permission but support different set of attributes for conditions. Thus, by using a suboperation, you can specify one condition for access to a subset of operations that support a given parameter. Then, you can use another access condition for operations with the same action that doesn't support that parameter.

For example, the `Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write` action is required for over a dozen different service operations. Some of these operations can accept blob index tags as request parameter, while others don't. For operations that accept blob index tags as a parameter, you can use blob index tags in a Request condition. However, if such a condition is defined on the `Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write` action, all operations that don't accept tags as a request parameter cannot evaluate this condition, and will fail the authorization access check.

In this case, the optional suboperation `Blob.WriteWithTagHeaders` can be used to apply a condition to only those operations that support blob index tags as a request parameter.

## NOTE

Blobs also support the ability to store arbitrary user-defined key-value metadata. Although metadata is similar to blob index tags, you must use blob index tags with conditions. For more information, see [Manage and find Azure Blob data with blob index tags](#).

In this preview, storage accounts support the following suboperations:

DISPLAY NAME	DATAACTION	SUBOPERATION
List blobs	<code>Microsoft.Storage/storageAccounts/blobContainers/blobs/read</code>	
Read a blob	<code>Microsoft.Storage/storageAccounts/blobContainers/blobs/read</code>	<small>NOT</small>
Read content from a blob with tag conditions	<code>Microsoft.Storage/storageAccounts/blobContainers/blobs/read</code>	<code>Blob.ReadWithTagConditions (deprecated)</code>
Sets the access tier on a blob	<code>Microsoft.Storage/storageAccounts/blobContainers/blobs/write</code>	
Write to a blob with blob index tags	<code>Microsoft.Storage/storageAccounts/blobContainers/blobs/write</code>	<code>Microsoft.Storage/storageAccounts/blobContainers/blobs/write</code>

## Azure Blob storage actions and suboperations

This section lists the supported Azure Blob storage actions and suboperations you can target for conditions.

### List blobs

PROPERTY	VALUE
Display name	List blobs
Description	List blobs operation.

PROPERTY	VALUE
DataAction	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read
Suboperation	Blob.List
Resource attributes	Account name Is hierarchical namespace enabled Container name
Request attributes	Blob prefix
Principal attributes support	True
Examples	<pre>! (ActionMatches('Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read') AND SubOperationMatches('Blob.List'))</pre> <p>Example: Read or list blobs in named containers with a path</p>

#### Read a blob

PROPERTY	VALUE
Display name	Read a blob
Description	All blob read operations excluding list.
DataAction	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read
Suboperation	NOT Blob.List
Resource attributes	Account name Is Current Version Is hierarchical namespace enabled Container name Blob path Encryption scope name
Request attributes	Version ID Snapshot
Principal attributes support	True
Examples	<pre>! (ActionMatches('Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read') AND NOT SubOperationMatches('Blob.List'))</pre> <p>Example: Read blobs in named containers with a path</p>

#### Read content from a blob with tag conditions

##### IMPORTANT

Although [Read content from a blob with tag conditions](#) is currently supported for compatibility with conditions implemented during the ABAC feature preview, that suboperation has been deprecated and Microsoft recommends using the "Read a blob" action instead.

When configuring ABAC conditions in the Azure portal, you might see "DEPRECATED: Read content from a blob with tag conditions". Remove the operation and replace it with the "Read a blob" operation instead.

If you are authoring your own condition where you want to restrict read access by tag conditions, please refer to [Example: Read blobs with a blob index tag](#).

#### Read blob index tags

PROPERTY	VALUE
Display name	Read blob index tags
Description	DataAction for reading blob index tags.
DataAction	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/read
Suboperation	

PROPERTY	VALUE
Resource attributes	Account name Is Current Version Is hierarchical namespace enabled Container name Blob path Blob index tags [Values in key] Blob index tags [Keys]
Request attributes	Version ID Snapshot
Principal attributes support	True
Learn more	<a href="#">Manage and find Azure Blob data with blob index tags</a>

#### Find blobs by tags

PROPERTY	VALUE
Display name	Find blobs by tags
Description	DataAction for finding blobs by index tags.
DataAction	<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/filter/action</code>
Suboperation	
Resource attributes	Account name Is hierarchical namespace enabled
Request attributes	
Principal attributes support	True

#### Write to a blob

PROPERTY	VALUE
Display name	Write to a blob
Description	DataAction for writing to blobs.
DataAction	<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write</code>
Suboperation	
Resource attributes	Account name Is hierarchical namespace enabled Container name Blob path Encryption scope name
Request attributes	
Principal attributes support	True
Examples	<pre>!(ActionMatches('Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write'))</pre> <p>Example: <a href="#">Read, write, or delete blobs in named containers</a></p>

#### Sets the access tier on a blob

PROPERTY	VALUE
Display name	Sets the access tier on a blob
Description	DataAction for writing to blobs.
DataAction	<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write</code>
Suboperation	<code>Blob.Write.Tier</code>

PROPERTY	VALUE
Resource attributes	Account name Is Current Version Is hierarchical namespace enabled Container name Blob path Encryption scope name
Request attributes	Version ID Snapshot
Principal attributes support	True
Examples	! (ActionMatches('Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write' AND SubOperationMatches('Blob.Write.Tier'))

#### Write to a blob with blob index tags

PROPERTY	VALUE
Display name	Write to a blob with blob index tags
Description	REST operations: Put Blob, Put Block List, Copy Blob and Copy Blob From URL.
DataAction	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action
Suboperation	Blob.Write.WithTagHeaders
Resource attributes	Account name Is hierarchical namespace enabled Container name Blob path Encryption scope name
Request attributes	Blob index tags [Values in key] Blob index tags [Keys]
Principal attributes support	True
Examples	! (ActionMatches('Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write' AND SubOperationMatches('Blob.Write.WithTagHeaders')) ! (ActionMatches('Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action' AND SubOperationMatches('Blob.Write.WithTagHeaders')) Example: New blobs must include a blob index tag
Learn more	<a href="#">Manage and find Azure Blob data with blob index tags</a>

#### Create a blob or snapshot, or append data

PROPERTY	VALUE
Display name	Create a blob or snapshot, or append data
Description	DataAction for creating blobs.
DataAction	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action
Suboperation	
Resource attributes	Account name Is hierarchical namespace enabled Container name Blob path Encryption scope name
Request attributes	
Principal attributes support	True
Examples	! (ActionMatches('Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action')) Example: Read, write, or delete blobs in named containers

### Write blob index tags

PROPERTY	VALUE
Display name	Write blob index tags
Description	DataAction for writing blob index tags.
DataAction	<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write</code>
Suboperation	
Resource attributes	Account name Is Current Version Is hierarchical namespace enabled Container name Blob path Blob index tags [Values in key] Blob index tags [Keys]
Request attributes	Blob index tags [Values in key] Blob index tags [Keys] Version ID Snapshot
Principal attributes support	True
Examples	<pre>! (ActionMatches('Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write'))</pre> <p>Example: Existing blobs must have blob index tag keys</p>
Learn more	<a href="#">Manage and find Azure Blob data with blob index tags</a>

### Write Blob legal hold and immutability policy

PROPERTY	VALUE
Display name	Write Blob legal hold and immutability policy
Description	DataAction for writing Blob legal hold and immutability policy.
DataAction	<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/immutableStorage/runAs</code>
Suboperation	
Resource attributes	Account name Is hierarchical namespace enabled Container name Blob path
Request attributes	
Principal attributes support	True

### Delete a blob

PROPERTY	VALUE
Display name	Delete a blob
Description	DataAction for deleting blobs.
DataAction	<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/delete</code>
Suboperation	
Resource attributes	Account name Is Current Version Is hierarchical namespace enabled Container name Blob path
Request attributes	Version ID Snapshot
Principal attributes support	True

PROPERTY	VALUE
Examples	<pre>! (ActionMatches('Microsoft.Storage/storageAccounts/blobServices/containers/blobs/delete'))</pre> <p>Example: Read, write, or delete blobs in named containers</p>

#### Delete a version of a blob

PROPERTY	VALUE
Display name	Delete a version of a blob
Description	DataAction for deleting a version of a blob.
DataAction	<a href="#">Microsoft.Storage/storageAccounts/blobServices/containers/blobs/deleteBlobVersion/action</a>
Suboperation	
Resource attributes	Account name Is hierarchical namespace enabled Container name Blob path
Request attributes	Version ID
Principal attributes support	True
Examples	<pre>! (ActionMatches('Microsoft.Storage/storageAccounts/blobServices/containers/blobs/delete'))</pre> <p>Example: Delete old blob versions</p>

#### Permanently delete a blob overriding soft-delete

PROPERTY	VALUE
Display name	Permanently delete a blob overriding soft-delete
Description	DataAction for permanently deleting a blob overriding soft-delete.
DataAction	<a href="#">Microsoft.Storage/storageAccounts/blobServices/containers/blobs/permanentDelete/action</a>
Suboperation	
Resource attributes	Account name Is Current Version Is hierarchical namespace enabled Container name Blob path
Request attributes	Version ID Snapshot
Principal attributes support	True

#### Modify permissions of a blob

PROPERTY	VALUE
Display name	Modify permissions of a blob
Description	DataAction for modifying permissions of a blob.
DataAction	<a href="#">Microsoft.Storage/storageAccounts/blobServices/containers/blobs/modifyPermissions/action</a>
Suboperation	
Resource attributes	Account name Is hierarchical namespace enabled Container name Blob path
Request attributes	
Principal attributes support	True

#### Change ownership of a blob

PROPERTY	VALUE
Display name	Change ownership of a blob
Description	DataAction for changing ownership of a blob.
DataAction	<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/manageOwnership/action</code>
Suboperation	
Resource attributes	Account name Is hierarchical namespace enabled Container name Blob path
Request attributes	
Principal attributes support	True

#### Rename a file or a directory

PROPERTY	VALUE
Display name	Rename a file or a directory
Description	DataAction for renaming files or directories.
DataAction	<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/move/action</code>
Suboperation	
Resource attributes	Account name Is hierarchical namespace enabled Container name Blob path
Request attributes	
Principal attributes support	True

#### All data operations for accounts with hierarchical namespace enabled

PROPERTY	VALUE
Display name	All data operations for accounts with hierarchical namespace enabled
Description	DataAction for all data operations on storage accounts with hierarchical namespace enabled. If your role definition includes the <code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/runAsSuperUser/action</code> action, you should target this action in your condition. Targeting this action ensures the condition will still work as expected if hierarchical namespace is enabled for a storage account.
DataAction	<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/runAsSuperUser/action</code>
Suboperation	
Resource attributes	Account name Is Current Version Is hierarchical namespace enabled Container name Blob path
Request attributes	
Principal attributes support	True

PROPERTY	VALUE
Examples	Example: Read, write, or delete blobs in named containers Example: Read blobs in named containers with a path Example: Read or list blobs in named containers with a path Example: Write blobs in named containers with a path Example: Read only current blob versions Example: Read current blob versions and any blob snapshots Example: Read only storage accounts with hierarchical namespace enabled
Learn more	<a href="#">Azure Data Lake Storage Gen2 hierarchical namespace</a>

## Azure Queue storage actions

This section lists the supported Azure Queue storage actions you can target for conditions.

### Peek messages

PROPERTY	VALUE
Display name	Peek messages
Description	DataAction for peeking messages.
DataAction	<code>Microsoft.Storage/storageAccounts/queueServices/queues/messages/read</code>
Resource attributes	Account name Queue name
Request attributes	
Principal attributes support	True

### Put a message

PROPERTY	VALUE
Display name	Put a message
Description	DataAction for putting a message.
DataAction	<code>Microsoft.Storage/storageAccounts/queueServices/queues/messages/add/action</code>
Resource attributes	Account name Queue name
Request attributes	
Principal attributes support	True

### Put or update a message

PROPERTY	VALUE
Display name	Put or update a message
Description	DataAction for putting or updating a message.
DataAction	<code>Microsoft.Storage/storageAccounts/queueServices/queues/messages/write</code>
Resource attributes	Account name Queue name
Request attributes	
Principal attributes support	True

### Clear messages

PROPERTY	VALUE
Display name	Clear messages
Description	DataAction for clearing messages.

PROPERTY	VALUE
DataAction	Microsoft.Storage/storageAccounts/queueServices/queues/messages/delete
Resource attributes	Account name Queue name
Request attributes	
Principal attributes support	True

#### Get or delete messages

PROPERTY	VALUE
Display name	Get or delete messages
Description	DataAction for getting or deleting messages.
DataAction	Microsoft.Storage/storageAccounts/queueServices/queues/messages/process/action
Resource attributes	Account name Queue name
Request attributes	
Principal attributes support	True

## Azure Blob storage attributes

This section lists the Azure Blob storage attributes you can use in your condition expressions depending on the action you target. If you select multiple actions for a single condition, there might be fewer attributes to choose from for your condition because the attributes must be available across the selected actions.

#### NOTE

Attributes and values listed are considered case-insensitive, unless stated otherwise.

#### Account name

PROPERTY	VALUE
Display name	Account name
Description	Name of a storage account.
Attribute	Microsoft.Storage/storageAccounts:name
Attribute source	Resource
Attribute type	String
Examples	<pre>@Resource[Microsoft.Storage/storageAccounts:name] StringEquals 'sampleaccount'</pre> <p>Example: Read or write blobs in named storage account with specific encryption scope</p>

#### Blob index tags [Keys]

PROPERTY	VALUE
Display name	Blob index tags [Keys]
Description	Index tags on a blob resource. Arbitrary user-defined key-value properties that you can store alongside a blob resource. Use when you want to check the key in blob index tags.
Attribute	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags\$keys\$&
Attribute source	Resource Request
Attribute type	StringList

PROPERTY	VALUE
Is key case sensitive	True
Hierarchical namespace support	False
Examples	<pre>@Request[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags&amp;\$keys\$&amp;] ForAllOfAnyValues:StringEquals {'Project', 'Program'}</pre> <p>Example: Existing blobs must have blob index tag keys</p>
Learn more	<a href="#">Manage and find Azure Blob data with blob index tags</a> <a href="#">Azure Data Lake Storage Gen2 hierarchical namespace</a>

### Blob index tags [Values in key]

PROPERTY	VALUE
Display name	Blob index tags [Values in key]
Description	Index tags on a blob resource. Arbitrary user-defined key-value properties that you can store alongside a blob resource. Use when you want to check both the key (case-sensitive) and value in blob index tags.
Attribute	<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags</code>
Attribute source	Resource Request
Attribute type	String
Is key case sensitive	True
Hierarchical namespace support	False
Examples	<pre>@Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags: keyname &lt;\$key_case_sensitive\$&gt; @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Project StringEquals 'Cascade'</pre> <p>Example: Read blobs with a blob index tag</p>
Learn more	<a href="#">Manage and find Azure Blob data with blob index tags</a> <a href="#">Azure Data Lake Storage Gen2 hierarchical namespace</a>

### Blob path

PROPERTY	VALUE
Display name	Blob path
Description	Path of a virtual directory, blob, folder or file resource. Use when you want to check the blob name or folders in a blob path.
Attribute	<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs:path</code>
Attribute source	Resource
Attribute type	String
Examples	<pre>@Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:path] StringLike 'readonly/*'</pre> <p>Example: Read blobs in named containers with a path</p>

#### NOTE

When specifying conditions for the `Microsoft.Storage/storageAccounts/blobServices/containers/blobs:path` attribute, the values shouldn't include the container name or a preceding slash (`/`) character. Use the path characters without any URL encoding.

### Blob prefix

PROPERTY	VALUE
Display name	Blob prefix

PROPERTY	VALUE
Description	Allowed prefix of blobs to be listed. Path of a virtual directory or folder resource. Use when you want to check the folders in a blob path.
Attribute	<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs:prefix</code>
Attribute source	Request
Attribute type	String
Examples	<pre>@Request[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:prefix] StringStartsWith 'readonly/'</pre> <p>Example: Read or list blobs in named containers with a path</p>

#### NOTE

When specifying conditions for the `Microsoft.Storage/storageAccounts/blobServices/containers/blobs:prefix` attribute, the values shouldn't include the container name or a preceding slash (`/`) character. Use the path characters without any URL encoding.

#### Container name

PROPERTY	VALUE
Display name	Container name
Description	Name of a storage container or file system. Use when you want to check the container name.
Attribute	<code>Microsoft.Storage/storageAccounts/blobServices/containers:name</code>
Attribute source	Resource
Attribute type	String
Examples	<pre>@Resource[Microsoft.Storage/storageAccounts/blobServices/containers:name] StringEquals 'blobs-example-container'</pre> <p>Example: Read, write, or delete blobs in named containers</p>

#### Encryption scope name

PROPERTY	VALUE
Display name	Encryption scope name
Description	Name of the encryption scope used to encrypt data. Available only for storage accounts where hierarchical namespace is not enabled.
Attribute	<code>Microsoft.Storage/storageAccounts/encryptionScopes:name</code>
Attribute source	Resource
Attribute type	String
Exists support	True
Examples	<pre>@Resource[Microsoft.Storage/storageAccounts/encryptionScopes:name] ForAnyOfAnyValues:StringEquals {'validScope1', 'validScope2'}</pre> <p>Example: Read blobs with specific encryption scopes</p>
Learn more	<a href="#">Create and manage encryption scopes</a>

#### Is Current Version

PROPERTY	VALUE
Display name	Is Current Version
Description	Identifies if the resource is the current version of the blob, in contrast of a snapshot or a specific blob version.
Attribute	<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs:isCurrentVersion</code>

PROPERTY	VALUE
Attribute source	Resource
Attribute type	Boolean
Examples	<pre>@Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:isCurrentVersion] BoolEquals true</pre> <p>Example: Read only current blob versions Example: Read current blob versions and a specific blob version</p>

#### Is hierarchical namespace enabled

PROPERTY	VALUE
Display name	Is hierarchical namespace enabled
Description	Whether hierarchical namespace is enabled on the storage account. Applicable only at resource group scope or above.
Attribute	Microsoft.Storage/storageAccounts:isHnsEnabled
Attribute source	Resource
Attribute type	Boolean
Examples	<pre>@Resource[Microsoft.Storage/storageAccounts:isHnsEnabled] BoolEquals true</pre> <p>Example: Read only storage accounts with hierarchical namespace enabled</p>
Learn more	<a href="#">Azure Data Lake Storage Gen2 hierarchical namespace</a>

#### Snapshot

PROPERTY	VALUE
Display name	Snapshot
Description	The Snapshot identifier for the Blob snapshot. Available for storage accounts where hierarchical namespace is not enabled and currently in preview for storage accounts where hierarchical namespace is enabled.
Attribute	Microsoft.Storage/storageAccounts/blobServices/containers/blobs:snapshot
Attribute source	Request
Attribute type	DateTime
Exists support	True
Hierarchical namespace support	False
Examples	<pre>Exists @Request[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:snapshot]</pre> <p>Example: Read current blob versions and any blob snapshots</p>
Learn more	<a href="#">Blob snapshots</a> <a href="#">Azure Data Lake Storage Gen2 hierarchical namespace</a>

#### Version ID

PROPERTY	VALUE
Display name	Version ID
Description	The version ID of the versioned Blob. Available only for storage accounts where hierarchical namespace is not enabled.
Attribute	Microsoft.Storage/storageAccounts/blobServices/containers/blobs:versionId
Attribute source	Request

PROPERTY	VALUE
Attribute type	DateTime
Exists support	True
Hierarchical namespace support	False
Examples	<pre>@Request[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:versionId] DateTimeEquals '2022-06-01T23:38:32.8883645Z'</pre> <p>Example: Read current blob versions and a specific blob version  Example: Read current blob versions and any blob snapshots</p>
Learn more	<a href="#">Azure Data Lake Storage Gen2 hierarchical namespace</a>

## Azure Queue storage attributes

This section lists the Azure Queue storage attributes you can use in your condition expressions depending on the action you target.

### Queue name

PROPERTY	VALUE
Display name	Queue name
Description	Name of a storage queue.
Attribute	<code>Microsoft.Storage/storageAccounts/queueServices/queues:name</code>
Attribute source	Resource
Attribute type	String

## See also

- [Example Azure role assignment conditions \(preview\)](#)
- [Azure role assignment condition format and syntax \(preview\)](#)
- [Troubleshoot Azure role assignment conditions \(preview\)](#)

# Security considerations for Azure role assignment conditions in Azure Storage (preview)

8/22/2022 • 5 minutes to read • [Edit Online](#)

## IMPORTANT

Azure ABAC and Azure role assignment conditions are currently in preview. This preview version is provided without a service level agreement, and it is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

To fully secure resources using [Azure attribute-based access control \(Azure ABAC\)](#), you must also protect the [attributes](#) used in the [Azure role assignment conditions](#). For instance, if your condition is based on a file path, then you should beware that access can be compromised if the principal has an unrestricted permission to rename a file path.

This article describes security considerations that you should factor into your role assignment conditions.

## Use of other authorization mechanisms

Role assignment conditions are only evaluated when using Azure RBAC for authorization. These conditions can be bypassed if you allow access using alternate authorization methods:

- [Shared Key](#) authorization
- [Account shared access signature \(SAS\)](#)
- [Service SAS](#).

Similarly, conditions are not evaluated when access is granted using [access control lists \(ACLs\)](#) in storage accounts with a [hierarchical namespace](#) (HNS).

You can prevent shared key, account-level SAS, and service-level SAS authorization by [disabling shared key authorization](#) for your storage account. Since user delegation SAS depends on Azure RBAC, role-assignment conditions are evaluated when using this method of authorization.

## NOTE

Role-assignment conditions are not evaluated when access is granted using ACLs with Data Lake Storage Gen2. In this case, you must plan the scope of access so it does not overlap with that granted through ACLs.

## Securing storage attributes used in conditions

### Blob path

When using blob path as a `@Resource` attribute for a condition, you should also prevent users from renaming a blob to get access to a file when using accounts that have a hierarchical namespace. For example, if you want to author a condition based on blob path, you should also restrict the user's access to the following actions:

ACTION	DESCRIPTION
<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write</code>	This action allows customers to rename a file using the Path Create API.
<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write</code>	This action allows access to various file system and path operations.

## Blob index tags

[Blob index tags](#) are used as free-form attributes for conditions in storage. If you author any access conditions by using these tags, you must also protect the tags themselves. Specifically, the

`Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write` DataAction allows users to modify the tags on a storage object. You can restrict this action to prevent users from manipulating a tag key or value to gain access to unauthorized objects.

In addition, if blob index tags are used in conditions, data may be vulnerable if the data and the associated index tags are updated in separate operations. You can use `@Request` conditions on blob write operations to require that index tags be set in the same update operation. This approach can help secure data from the instant it's written to storage.

### Tags on copied blobs

By default, blob index tags are not copied from a source blob to the destination when you use [Copy Blob](#) API or any of its variants. To preserve the scope of access for blob upon copy, you should copy the tags as well.

### Tags on snapshots

Tags on blob snapshots cannot be modified. This implies that you must update the tags on a blob before taking the snapshot. If you modify the tags on a base blob, the tags on its snapshot will continue to have their previous value.

If a tag on a base blob is modified after a snapshot is taken, the scope of access may be different for the base blob and the snapshot.

### Tags on blob versions

Blob index tags aren't copied when a blob version is created through the [Put Blob](#), [Put Block List](#) or [Copy Blob](#) APIs. You can specify tags through the header for these APIs.

Tags can be set individually on a current base blob and on each blob version. When you modify tags on a base blob, the tags on previous versions are not updated. If you want to change the scope of access for a blob and all its versions using tags, you must update the tags on each version.

### Querying and filtering limitations for versions and snapshots

When using tags to query and filter blobs in a container, only the base blobs are included in the response. Blob versions or snapshots with the requested keys and values aren't included.

## Roles and permissions

If you're using role assignment conditions for [Azure built-in roles](#), you should carefully review all the permissions that the role grants to a principal.

### Inherited role assignments

Role assignments can be configured for a management group, subscription, resource group, storage account, or a container, and are inherited at each level in the stated order. Azure RBAC has an additive model, so the effective permissions are the sum of role assignments at each level. If a principal has the same permission assigned to them through multiple role assignments, then access for an operation using that permission is evaluated separately for each assignment at every level.

Since conditions are implemented as conditions on role assignments, any unconditional role assignment can allow users to bypass the condition. Let's say you assign the *Storage Blob Data Contributor* role to a user for a storage account and on a subscription, but add a condition only to the assignment for the storage account. In this case, the user will have unrestricted access to the storage account through the role assignment at the subscription level.

That's why you should apply conditions consistently for all role assignments across a resource hierarchy.

## Other considerations

### Condition operations that write blobs

Many operations that write blobs require either the

`Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write` or the

`Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action` permission. Built-in roles, such as *Storage Blob Data Owner* and *Storage Blob Data Contributor* grant both permissions to a security principal.

When you define a role assignment condition on these roles, you should use identical conditions on both these permissions to ensure consistent access restrictions for write operations.

### Behavior for Copy Blob and Copy Blob from URL

For the *Copy Blob* and *Copy Blob From URL* operations, `@Request` conditions using blob path as attribute on the

`Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write` action and its suboperations are evaluated only for the destination blob.

For conditions on the source blob, `@Resource` conditions on the

`Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/read` action are evaluated.

## See also

- [Authorize access to blobs using Azure role assignment conditions \(preview\)](#)
- [Actions and attributes for Azure role assignment conditions in Azure Storage \(preview\)](#)
- [What is Azure attribute-based access control \(Azure ABAC\)? \(preview\)](#)

# Example Azure role assignment conditions (preview)

8/22/2022 • 20 minutes to read • [Edit Online](#)

## IMPORTANT

Azure ABAC and Azure role assignment conditions are currently in preview. This preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

This article lists some examples of role assignment conditions.

## Prerequisites

For information about the prerequisites to add or edit role assignment conditions, see [Conditions prerequisites](#).

## Blob index tags

### IMPORTANT

Although "Read content from a blob with tag conditions" is currently supported for compatibility with conditions implemented during the ABAC feature preview, that suboperation has been deprecated and Microsoft recommends using the "Read a blob" suboperation instead.

When configuring ABAC conditions in the Azure portal, you might see "DEPRECATED: Read content from a blob with tag conditions". Remove the operation and replace it with the "Read a blob" suboperation instead.

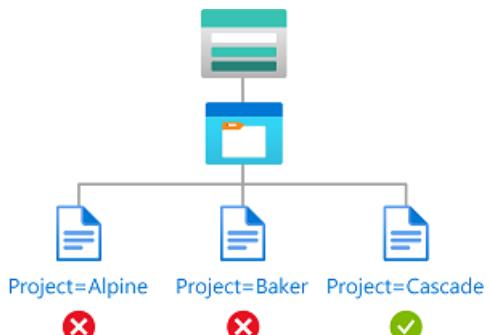
If you are authoring your own condition where you want to restrict read access by tag conditions, please refer to [Example: Read blobs with a blob index tag](#).

### Example: Read blobs with a blob index tag

This condition allows users to read blobs with a [blob index tag](#) key of Project and a value of Cascade. Attempts to access blobs without this key-value tag will not be allowed.

You must add this condition to any role assignments that include the following action.

ACTION	NOTES
<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read</code>	
<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read</code>	Add this condition to a role definition that includes this action, such as Storage Blob Data Owner.



```

(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'} AND NOT
SubOperationMatches{'Blob.List'})
)
OR
(
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Project<$key_case_sensitive$>
] StringEquals 'Cascade'
)
)

```

#### Azure portal

Here are the settings to add this condition using the Azure portal.

CONDITION #1	SETTING
Actions	<a href="#">Read a blob</a>
Attribute source	Resource
Attribute	<a href="#">Blob index tags [Values in key]</a>
Key	{keyName}
Operator	<a href="#">StringEquals</a>
Value	{keyValue}

**1. Add action \***

Click Add action to select the actions you want to allow if the condition is true. [Learn more](#)

+ Add action  Delete

Action Type	Action	Operation
Data Action	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read	Read a blob 

Select actions

**2. Build expression**

Build one or more expressions. If the expressions evaluate to true, access is allowed to the selected actions. [Learn more](#)

+ Add expression  Delete |  Group | 

1 @Resource[Microsoft.Storage.../tags:Project<\$key\_case\_sensitive\$>] StringEquals 'Cascade'

Attribute source  Resource	Operator  StringEquals	Value  Cascade
Attribute  Blob index tags [Values in key]	<input checked="" type="radio"/> Value <input type="radio"/> Attribute	
Key  Project		

Negate this expression

+ Add expression

+ Add condition

 Save  Discard

## Azure PowerShell

Here's how to add this condition using Azure PowerShell.

```
$condition = "((!(ActionMatches('Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read') AND NOT SubOperationMatches('Blob.List'))) OR (@Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Project<$key_case_sensitive`$>] StringEquals 'Cascade'))"
$testRa = Get-AzRoleAssignment -Scope $scope -RoleDefinitionName $roleDefinitionName -ObjectId $userObjectID
$testRa.Condition = $condition
$testRa.ConditionVersion = "2.0"
Set-AzRoleAssignment -InputObject $testRa -PassThru
```

Here's how to test this condition.

```
$bearerCtx = New-AzStorageContext -StorageAccountName $storageAccountName
Get-AzStorageBlob -Container <containerName> -Blob <blobName> -Context $bearerCtx
```

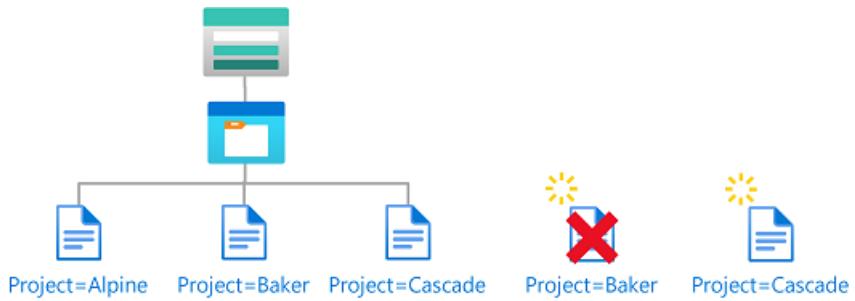
## Example: New blobs must include a blob index tag

This condition requires that any new blobs must include a [blob index tag](#) key of Project and a value of Cascade.

There are two actions that allow you to create new blobs, so you must target both. You must add this condition to any role assignments that include one of the following actions.

ACTION	NOTES
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write	
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action	

ACTION	NOTES
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write	Add this definition to your role definition if you want to include this action, such as Storage Blob Data Owner.



```

(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write'} AND
SubOperationMatches{'Blob.Write.WithTagHeaders'})
 AND
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action'} AND
SubOperationMatches{'Blob.Write.WithTagHeaders'})
)
OR
(
 @Request[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Project<$key_case_sensitive$>]
StringEquals 'Cascade'
)
)

```

### Azure portal

Here are the settings to add this condition using the Azure portal.

CONDITION #1	SETTING
Actions	Write to a blob with blob index tags Write to a blob with blob index tags
Attribute source	Request
Attribute	Blob index tags [Values in key]
Key	{keyName}
Operator	StringEquals
Value	{keyValue}

Got feedback?

Role: Storage Blob Data Owner

Editor type:  Visual  Code

Description: Add a friendly description for the condition.

**Condition #1**

1. Add action \*

Click Add action to select the actions you want to allow if the condition is true. [Learn more](#)

+ Add action  Delete

Action Type	Action	Operation
<input type="checkbox"/> Data Action	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write	Write to a blob with blob index tags <input type="radio"/>
<input type="checkbox"/> Data Action	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action	Write to a blob with blob index tags <input type="radio"/>

Select actions

2. Build expression

Build one or more expressions. If the expressions evaluate to true, access is allowed to the selected actions. [Learn more](#)

+ Add expression  Delete  Group  Ungroup

1 @Request[Microsoft.Storage/.../tags:Project<\$key\_case\_sensitive\$>] StringEquals 'Cascade'

Attribute source  Request  Blob index tags  Key

Operator  StringEquals  Value  Attribute

Value  Cascade

Negate this expression

Save  Discard

## Azure PowerShell

Here's how to add this condition using Azure PowerShell.

```
$condition = "((!(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write'}) AND SubOperationMatches{'Blob.Write.WithTagHeaders'}) AND ! (ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action'}) AND SubOperationMatches{'Blob.Write.WithTagHeaders'})) OR (@Request[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Project<$key_case_sensitive`$>] StringEquals 'Cascade'))"
$testRa = Get-AzRoleAssignment -Scope $scope -RoleDefinitionName $roleDefinitionName -ObjectId $userObjectID
$testRa.Condition = $condition
$testRa.ConditionVersion = "2.0"
Set-AzRoleAssignment -InputObject $testRa -PassThru
```

Here's how to test this condition.

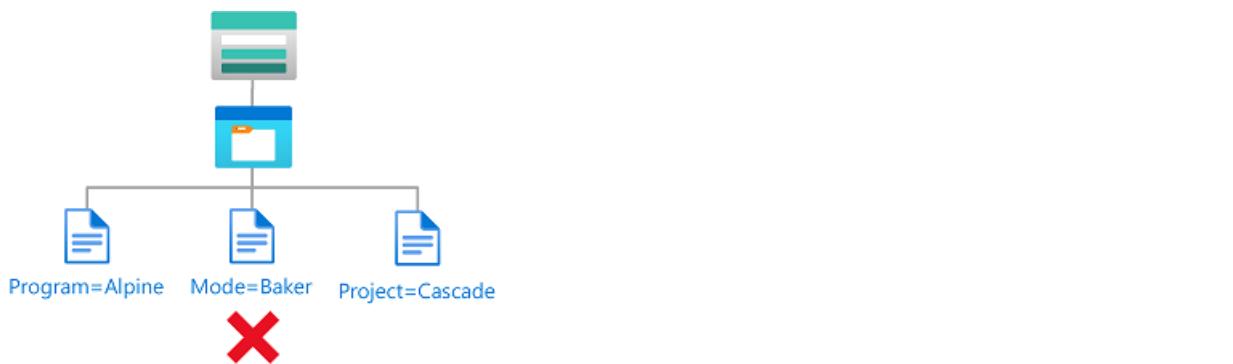
```
$localSrcFile = # path to an example file, can be an empty txt
$ungrantedTag = @{'Project'='Baker'}
$grantedTag = @{'Project'='Cascade'}
Get new context for request
$bearerCtx = New-AzStorageContext -StorageAccountName $storageAccountName
try ungranted tags
$content = Set-AzStorageBlobContent -File $localSrcFile -Container example2 -Blob "Example2.txt" -Tag $ungrantedTag -Context $bearerCtx
try granted tags
$content = Set-AzStorageBlobContent -File $localSrcFile -Container example2 -Blob "Example2.txt" -Tag $grantedTag -Context $bearerCtx
```

## Example: Existing blobs must have blob index tag keys

This condition requires that any existing blobs be tagged with at least one of the allowed [blob index tag](#) keys: Project or Program. This condition is useful for adding governance to existing blobs.

There are two actions that allow you to update tags on existing blobs, so you must target both. You must add this condition to any role assignments that include one of the following actions.

ACTION	NOTES
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write	
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write	
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write	Add this condition to any role assignment that includes this action, such as Storage Blob Data Owner.



```
(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write'}) AND
 SubOperationMatches{'Blob.WriteWithTagHeaders'})
 AND
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write'})
)
OR
(
 @Request[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags$keys$&]
 ForAllOfAnyValues:StringEquals {'Project', 'Program'}
)
)
```

## Azure portal

Here are the settings to add this condition using the Azure portal.

CONDITION #1	SETTING
Actions	<a href="#">Write to a blob with blob index tags</a> <a href="#">Write blob index tags</a>
Attribute source	Request
Attribute	<a href="#">Blob index tags [Keys]</a>
Operator	<a href="#">ForAllOfAnyValues:StringEquals</a>

CONDITION #1	SETTING
Value	{keyName1} {keyName2}

The screenshot shows the 'Add role assignment condition' dialog in the Azure portal. At the top, it displays the 'Role' as 'Storage Blob Data Owner'. Below that, the 'Editor type' is set to 'Visual'. A section titled 'Condition #1' is expanded, showing:

- 1. Add action \***: Click Add action to select the actions you want to allow if the condition is true. It lists two actions:
 

Action Type	Action	Operation
Data Action	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write	Write to a blob with blob index tags
Data Action	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags...	Write blob index tags
- 2. Build expression \***: Build one or more expressions for the action you selected. If the expressions evaluate to true, access is allowed to the selected action. It shows a complex expression involving 'Request' source, 'ForAllOfAnyValues:StringEquals' operator, and values 'Project' and 'Program'.

At the bottom, there are 'Save' and 'Discard' buttons.

## Azure PowerShell

Here's how to add this condition using Azure PowerShell.

```
$condition = "((!(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write'}) AND SubOperationMatches{'Blob.Write.WithTagHeaders'}) AND ! (ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write'})) OR (@Request[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags`$keys`$&] ForAllOfAnyValues:StringEquals {'Project', 'Program'}))"
$testRa = Get-AzRoleAssignment -Scope $scope -RoleDefinitionName $roleDefinitionName -ObjectId $userObjectID
$testRa.Condition = $condition
$testRa.ConditionVersion = "2.0"
Set-AzRoleAssignment -InputObject $testRa -PassThru
```

Here's how to test this condition.

```

$localSrcFile = # path to an example file, can be an empty txt
$ungrantedTag = @{'Mode'='Baker'}
$grantedTag = @{'Program'='Alpine';'Project'='Cascade'}
Get new context for request
$bearerCtx = New-AzStorageContext -StorageAccountName $storageAccountName
try ungranted tags
$content = Set-AzStorageBlobContent -File $localSrcFile -Container example3 -Blob "Example3.txt" -Tag
$ungrantedTag -Context $bearerCtx
try granted tags
$content = Set-AzStorageBlobContent -File $localSrcFile -Container example3 -Blob "Example3.txt" -Tag
$grantedTag -Context $bearerCtx

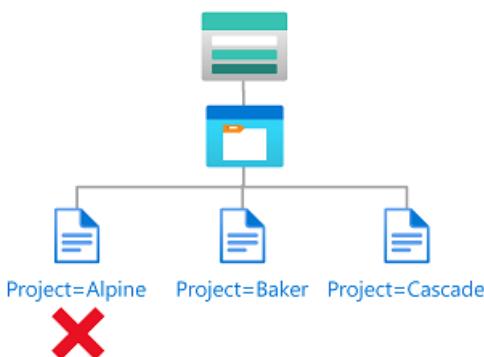
```

### Example: Existing blobs must have a blob index tag key and values

This condition requires that any existing blobs to have a [blob index tag](#) key of Project and values of Cascade, Baker, or Skagit. This condition is useful for adding governance to existing blobs.

There are two actions that allow you to update tags on existing blobs, so you must target both. You must add this condition to any role assignments that include one of the following actions.

ACTION	NOTES
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write	
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write	
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write	Add this condition only if your role definition includes this action, such as Storage Blob Data Owner.



```

(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write'}) AND
 SubOperationMatches{'Blob.Write.WithTagHeaders'})
 AND
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write'})
)
OR
(
 @Request[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:$keys$]
 ForAnyOfAnyValues:StringEquals {'Project'}
 AND

 @Request[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Project<$key_case_sensitive$>]
 ForAllOfAnyValues:StringEquals {'Cascade', 'Baker', 'Skagit'}
)
)

```

### Azure portal

Here are the settings to add this condition using the Azure portal.

CONDITION #1	SETTING
Actions	Write to a blob with blob index tags Write blob index tags
Attribute source	Request
Attribute	Blob index tags [Keys]
Operator	ForAnyOfAnyValues:StringEquals
Value	{keyName}
Operator	And
Expression 2	
Attribute source	Request
Attribute	Blob index tags [Values in key]
Key	{keyName}
Operator	ForAllOfAnyValues:StringEquals
Value	{keyValue1} {keyValue2} {keyValue3}

Home >

## Add role assignment condition

**Role** Storage Blob Data Owner [\(i\)](#)

**Editor type**  Visual  Code

**Condition #1** [\(i\)](#)

**1. Add action \***  
Click Add action to select the actions you want to allow if the condition is true. [Learn more](#)

Action Type	Action	Operation
Data Action	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write	Write to a blob with blob index tags <a href="#">(i)</a>
Data Action	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags...	Write blob index tags <a href="#">(i)</a>

**Select actions**

**2. Build expression \***  
Build one or more expressions for the action you selected. If the expressions evaluate to true, access is allowed to the selected action. [Learn more](#)

1 [@Request\[Microsoft.Storage/.../tags&\\$keys\\$&\] ForAnyOfAnyValues:StringEquals \('Project'\)](#) [\(i\)](#)

And  Or

2 [@Request\[Microsoft.Storage/.../tags:\\$key\\_case\\_sensitive\\$\] ForAllOfAnyValues:StringEquals \('Cascade', 'Baker', 'Skagit'\)](#) [\(i\)](#)

Attribute source <a href="#">(i)</a> <input type="button" value="Request"/>	Operator <a href="#">(i)</a> <input type="button" value="ForAllOfAnyValues:StringEquals"/>	Value Cascade <a href="#">(i)</a> Baker <a href="#">(i)</a> <input type="text" value="Skagit"/> <a href="#">(i)</a>
Attribute <a href="#">(i)</a> <input type="button" value="Blob index tags [Values in key]"/>	Key <a href="#">(i)</a> <input type="button" value="Project"/>	<input checked="" type="radio"/> Value <input type="radio"/> Attribute
<input type="checkbox"/> Negate this expression		

[Save](#) [Discard](#)

### Azure PowerShell

Here's how to add this condition using Azure PowerShell.

```
$condition = "((!(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write'}) AND SubOperationMatches{'Blob.Write.WithTagHeaders'}) AND ! (ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write'})) OR (@Request[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags`$keys`$&] ForAnyOfAnyValues:StringEquals {'Project'}) AND @Request[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:$key_case_sensitive$] ForAllOfAnyValues:StringEquals {'Cascade', 'Baker', 'Skagit'})"
$testRa = Get-AzRoleAssignment -Scope $scope -RoleDefinitionName $roleDefinitionName -ObjectId $userObjectID
$testRa.Condition = $condition
$testRa.ConditionVersion = "2.0"
Set-AzRoleAssignment -InputObject $testRa -PassThru
```

Here's how to test this condition.

```

$localSrcFile = <pathToLocalFile>
$ungrantedTag = @{'Project'='Alpine'}
$grantedTag1 = @{'Project'='Cascade'}
$grantedTag2 = @{'Project'='Baker'}
$grantedTag3 = @{'Project'='Skagit'}
Get new context for request
$bearerCtx = New-AzStorageContext -StorageAccountName $storageAccountName
try ungranted tags
Set-AzStorageBlobTag -Container example4 -Blob "Example4.txt" -Tag $ungrantedTag -Context $bearerCtx
try granted tags
Set-AzStorageBlobTag -Container example4 -Blob "Example4.txt" -Tag $grantedTag1 -Context $bearerCtx
Set-AzStorageBlobTag -Container example4 -Blob "Example4.txt" -Tag $grantedTag2 -Context $bearerCtx
Set-AzStorageBlobTag -Container example4 -Blob "Example4.txt" -Tag $grantedTag3 -Context $bearerCtx

```

## Blob container names or paths

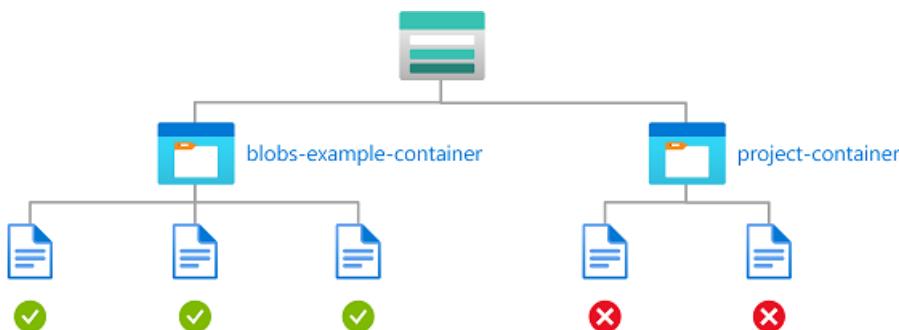
### Example: Read, write, or delete blobs in named containers

This condition allows users to read, write, or delete blobs in storage containers named blobs-example-container. This condition is useful for sharing specific storage containers with other users in a subscription.

There are five actions for read, write, and delete of existing blobs. You must add this condition to any role assignments that include one of the following actions.

ACTION	NOTES
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/delete	
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read	
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write	
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action	
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/*	Add if role definition includes this action, such as Storage Blob Data Owner. Add if the storage accounts included in this condition have hierarchical namespace enabled or might be enabled in the future.

Suboperations are not used in this condition because the subOperation is needed only when conditions are authored based on tags.



Storage Blob Data Owner

```

(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/delete'})
 AND
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'})
 AND
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write'})
 AND
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action'})
 AND
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/runAsSuperUser/action'})
)
OR
(
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers:name] StringEquals 'blobs-example-
container'
)
)

```

## Storage Blob Data Contributor

```

(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/delete'})
 AND
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'})
 AND
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write'})
 AND
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action'})
)
OR
(
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers:name] StringEquals 'blobs-example-
container'
)
)
```

### Azure portal

Here are the settings to add this condition using the Azure portal.

CONDITION #1	SETTING
Actions	<a href="#">Delete a blob</a> <a href="#">Read a blob</a> <a href="#">Write to a blob</a> <a href="#">Create a blob or snapshot, or append data</a> <a href="#">All data operations for accounts with hierarchical namespace enabled (if applicable)</a>
Attribute source	Resource
Attribute	<a href="#">Container name</a>
Operator	<a href="#">StringEquals</a>
Value	{containerName}

Home >

## Add role assignment condition

**Role** Storage Blob Data Owner [\(i\)](#)

**Editor type**  Visual  Code

**Condition #1** [\(i\)](#)

**1. Add action \***  
Click Add action to select the actions you want to allow if the condition is true. [Learn more](#)

Action Type	Action	Operation
Data Action	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/delete	Delete a blob <a href="#">(i)</a>
Data Action	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read	Read a blob <a href="#">(i)</a>
Data Action	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write	Write to a blob <a href="#">(i)</a>
Data Action	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add...	Create a blob or snapshot, or append data <a href="#">(i)</a>

**Select actions**

**2. Build expression \***  
Build one or more expressions for the action you selected. If the expressions evaluate to true, access is allowed to the selected action. [Learn more](#)

1 [@Resource\[Microsoft.Storage/.../containers:name\] StringEquals 'blobs-example-container'](#) [\(i\)](#)

Attribute source <a href="#">(i)</a> <input type="button" value="Resource"/>	Operator <a href="#">(i)</a> <input type="button" value="StringEquals"/>	Value <a href="#">(i)</a> <input type="button" value="blobs-example-container"/>
Attribute <a href="#">(i) <input type="button" value="Container name"/></a>	<input checked="" type="radio"/> Value <input type="radio"/> Attribute	
<input type="checkbox"/> Negate this expression		

**Save** **Discard**

### Azure PowerShell

Here's how to add this condition using Azure PowerShell.

```
$condition = "((!(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/delete'}))
AND !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'})) AND !
(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write'})) AND !
(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action'})) AND !
(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/runAsSuperUser/action'})) OR
(@Resource[Microsoft.Storage/storageAccounts/blobServices/containers:name] StringEquals 'blobs-example-
container'))"
$testRa = Get-AzRoleAssignment -Scope $scope -RoleDefinitionName $roleDefinitionName -ObjectId $userObjectID
$testRa.Condition = $condition
$testRa.ConditionVersion = "2.0"
Set-AzRoleAssignment -InputObject $testRa -PassThru
```

Here's how to test this condition.

```

$localSrcFile = <pathToLocalFile>
$grantedContainer = "blobs-example-container"
$ungrantedContainer = "ungranted"
Get new context for request
$bearerCtx = New-AzStorageContext -StorageAccountName $storageAccountName
Ungranted Container actions
$content = Set-AzStorageBlobContent -File $localSrcFile -Container $ungrantedContainer -Blob "Example5.txt" -Context $bearerCtx
$content = Get-AzStorageBlobContent -Container $ungrantedContainer -Blob "Example5.txt" -Context $bearerCtx
$content = Remove-AzStorageBlob -Container $ungrantedContainer -Blob "Example5.txt" -Context $bearerCtx
Granted Container actions
$content = Set-AzStorageBlobContent -File $localSrcFile -Container $grantedContainer -Blob "Example5.txt" -Context $bearerCtx
$content = Get-AzStorageBlobContent -Container $grantedContainer -Blob "Example5.txt" -Context $bearerCtx
$content = Remove-AzStorageBlob -Container $grantedContainer -Blob "Example5.txt" -Context $bearerCtx

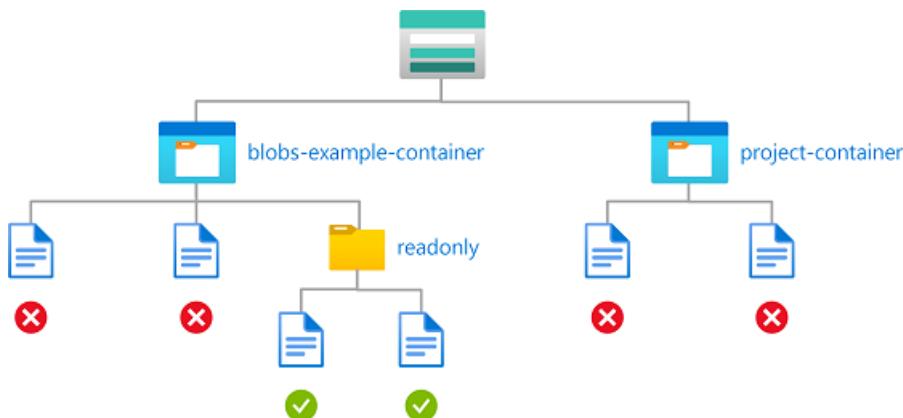
```

### Example: Read blobs in named containers with a path

This condition allows read access to storage containers named blobs-example-container with a blob path of readonly/\*. This condition is useful for sharing specific parts of storage containers for read access with other users in the subscription.

You must add this condition to any role assignments that include the following actions.

ACTION	NOTES
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read	
Microsoft.Storage/storageAccounts/blobServices/containers/readonly/*	Add if role definition includes this action, such as Storage Blob Data Owner. Add if the storage accounts included in this condition have hierarchical namespace enabled or might be enabled in the future.



Storage Blob Data Owner

```

(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'} AND NOT
SubOperationMatches{'Blob.List'})
 AND
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/runAsSuperUser/action'})
)
OR
(
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers:name] StringEquals 'blobs-example-
container'
 AND
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:path] StringLike 'readonly/*'
)
)

```

## Storage Blob Data Reader, Storage Blob Data Contributor

```

(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'} AND NOT
SubOperationMatches{'Blob.List'})
)
OR
(
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers:name] StringEquals 'blobs-example-
container'
 AND
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:path] StringLike 'readonly/*'
)
)

```

### Azure portal

Here are the settings to add this condition using the Azure portal.

CONDITION #1	SETTING
Actions	Read a blob All data operations for accounts with hierarchical namespace enabled (if applicable)
Attribute source	Resource
Attribute	Container name
Operator	StringEquals
Value	{containerName}
Expression 2	
Operator	And
Attribute source	Resource
Attribute	Blob path

CONDITION #1	SETTING
Operator	StringLike
Value	{pathString}

Condition #1

Action Type	Action	Operation
Data Action	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read	Read a blob

Condition #1

1. Add action \*

Click Add action to select the actions you want to allow if the condition is true. [Learn more](#)

Action Type	Action	Operation
Data Action	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read	Read a blob

Select actions

2. Build expression \*

Build one or more expressions for the action you selected. If the expressions evaluate to true, access is allowed to the selected action. [Learn more](#)

1 @Resource[Microsoft.Storage/.../blobServices/containers/blobs/path] StringEquals 'blobs-example-container'

2 @Resource[Microsoft.Storage/.../blobServices/containers/blobs/path] StringLike 'readonly/\*'

Attribute source: Resource  
Attribute: Blob path  
Operator: StringLike  
Value: readonly/\*

Negate this expression

Save Discard

## Azure PowerShell

Here's how to add this condition using Azure PowerShell.

```
$condition = "((!(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'}) AND NOT SubOperationMatches{'Blob.List'}) AND !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/runAsSuperUser/action'})) OR (@Resource[Microsoft.Storage/storageAccounts/blobServices/containers:name] StringEquals 'blobs-example-container' AND @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:path] StringLike 'readonly/*'))"
$testRa = Get-AzRoleAssignment -Scope $scope -RoleDefinitionName $roleDefinitionName -ObjectId $userObjectID
$testRa.Condition = $condition
$testRa.ConditionVersion = "2.0"
Set-AzRoleAssignment -InputObject $testRa -PassThru
```

Here's how to test this condition.

```

$grantedContainer = "blobs-example-container"
Get new context for request
$bearerCtx = New-AzStorageContext -StorageAccountName $storageAccountName
Try to get ungranted blob
$content = Get-AzStorageBlobContent -Container $grantedContainer -Blob "Ungranted.txt" -Context $bearerCtx
Try to get granted blob
$content = Get-AzStorageBlobContent -Container $grantedContainer -Blob "readonly/Example6.txt" -Context
$bearerCtx

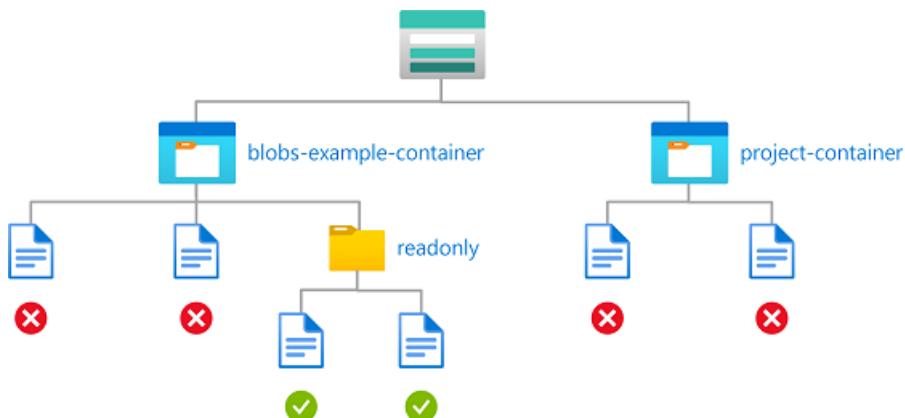
```

### Example: Read or list blobs in named containers with a path

This condition allows read access and also list access to storage containers named blobs-example-container with a blob path of readonly/\*. Condition #1 applies to read actions excluding list blobs. Condition #2 applies to list blobs. This condition is useful for sharing specific parts of storage containers for read or list access with other users in the subscription.

You must add this condition to any role assignments that include the following actions.

ACTION	NOTES
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read	
Microsoft.Storage/storageAccounts/blobServices/containers/read	Add if role definition includes this action, such as Storage Blob Data Owner. Add if the storage accounts included in this condition have hierarchical namespace enabled or might be enabled in the future.



Storage Blob Data Owner

```
(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'} AND NOT
SubOperationMatches{'Blob.List'})
 AND
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/runAsSuperUser/action'})
)
OR
(
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers:name] StringEquals 'blobs-example-
container'
 AND
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:path] StringStartsWith
'readonly/'
)
)
AND
(
 (
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'} AND
SubOperationMatches{'Blob.List'})
 AND
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/runAsSuperUser/action'})
)
OR
(
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers:name] StringEquals 'blobs-example-
container'
 AND
 @Request[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:prefix] StringStartsWith
'readonly/'
)
)
```

Storage Blob Data Reader, Storage Blob Data Contributor

```

(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'} AND NOT
SubOperationMatches{'Blob.List'})
)
OR
(
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers:name] StringEquals 'blobs-example-
container'
 AND
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:path] StringStartsWith
'readonly/'
)
)
AND
(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'} AND
SubOperationMatches{'Blob.List'})
)
OR
(
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers:name] StringEquals 'blobs-example-
container'
 AND
 @Request[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:prefix] StringStartsWith
'readonly/'
)
)
)

```

## Azure portal

Here are the settings to add this condition using the Azure portal.

### NOTE

The Azure portal uses prefix=" to list blobs from container's root directory. After the condition is added with the list blobs operation using prefix StringStartsWith 'readonly/', targeted users won't be able to list blobs from container's root directory in the Azure portal.

CONDITION #1	SETTING
Actions	<a href="#">Read a blob</a> <a href="#">All data operations for accounts with hierarchical namespace enabled</a> (if applicable)
Attribute source	Resource
Attribute	<a href="#">Container name</a>
Operator	<a href="#">StringEquals</a>
Value	{containerName}
Expression 2	
Operator	And
Attribute source	Resource

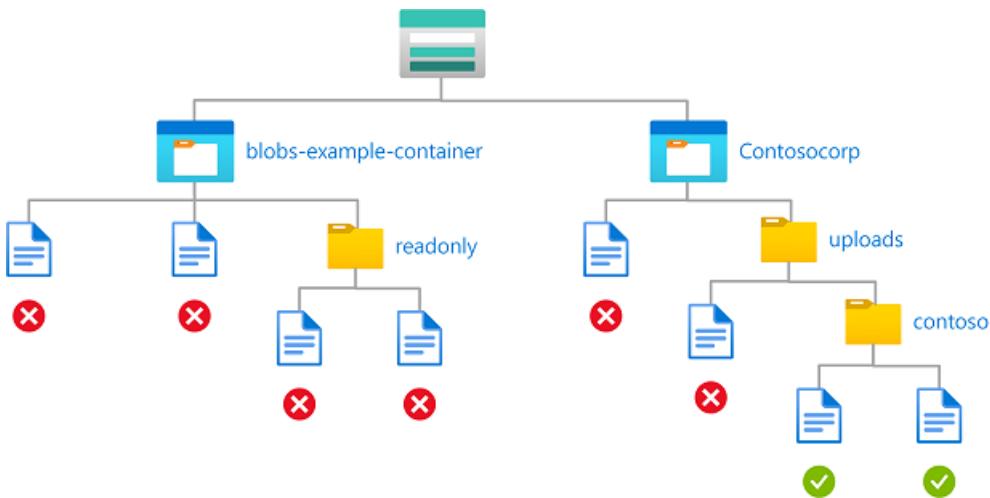
CONDITION #1	SETTING
Attribute	<a href="#">Blob path</a>
Operator	<a href="#">StringStartsWith</a>
Value	{pathString}
CONDITION #2	SETTING
Actions	<a href="#">List blobs</a> <a href="#">All data operations for accounts with hierarchical namespace enabled</a> (if applicable)
Attribute source	Resource
Attribute	<a href="#">Container name</a>
Operator	<a href="#">StringEquals</a>
Value	{containerName}
Expression 2	
Operator	And
Attribute source	Request
Attribute	<a href="#">Blob prefix</a>
Operator	<a href="#">StringStartsWith</a>
Value	{pathString}

#### Example: Write blobs in named containers with a path

This condition allows a partner (an Azure AD guest user) to drop files into storage containers named Contosocorp with a path of uploads/contoso/\*. This condition is useful for allowing other users to put data in storage containers.

You must add this condition to any role assignments that include the following actions.

ACTION	NOTES
<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write</code>	
<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action</code>	
<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write</code>	Add if role definition includes this action, such as Storage Blob Data Owner. Add if the storage accounts included in this condition have hierarchical namespace enabled or might be enabled in the future.



### Storage Blob Data Owner

```

(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write'})
 AND
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action'})
 AND
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/runAsSuperUser/action'})
)
OR
(
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers:name] StringEquals 'contosocorp'
 AND
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:path] StringLike
 'uploads/contoso/*'
)
)

```

### Storage Blob Data Contributor

```

(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write'})
 AND
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action'})
)
OR
(
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers:name] StringEquals 'contosocorp'
 AND
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:path] StringLike
 'uploads/contoso/*'
)
)

```

### Azure portal

Here are the settings to add this condition using the Azure portal.

CONDITION #1	SETTING
Actions	<p><a href="#">Write to a blob</a></p> <p><a href="#">Create a blob or snapshot, or append data</a></p> <p><a href="#">All data operations for accounts with hierarchical namespace enabled (if applicable)</a></p>

CONDITION #1	SETTING
Attribute source	Resource
Attribute	Container name
Operator	StringEquals
Value	{containerName}
<b>Expression 2</b>	
Operator	And
Attribute source	Resource
Attribute	Blob path
Operator	StringLike
Value	{pathString}

Home >  Add role assignment condition ... X

**Role**: Storage Blob Data Owner (i)

**Editor type**:  Visual  Code

**Condition #1** 

**1. Add action \***  
Click Add action to select the actions you want to allow if the condition is true. [Learn more](#)

Action Type	Action	Operation
Data Action	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write	Write to a blob <span style="color: blue;">(i)</span>
Data Action	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add...	Create a blob or snapshot, or append data <span style="color: blue;">(i)</span>

[Select actions](#)

**2. Build expression \***  
Build one or more expressions for the action you selected. If the expressions evaluate to true, access is allowed to the selected action. [Learn more](#)

1  `@Resource[Microsoft.Storage/.../containers:name] StringEquals 'contosocorp'`

And  Or

2  `@Resource[Microsoft.Storage/.../blobs:path] StringLike 'uploads/contoso/*'`

Attribute source <span style="color: blue;">(i)</span> <input type="button" value="Resource"/>	Operator <span style="color: blue;">(i)</span> <input type="button" value="StringLike"/>	Value <span style="color: blue;">(i)</span> <input type="text" value="uploads/contoso/*"/>
Attribute <span style="color: blue;">(i)</span> <input type="button" value="Blob path"/>	<input checked="" type="radio"/> Value <input type="radio"/> Attribute	
<input type="checkbox"/> Negate this expression		

 Save  Discard

## Azure PowerShell

Here's how to add this condition using Azure PowerShell.

```

$condition = "((!(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write'}))
AND !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action'})) AND !(
ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/runAsSuperUser/action'})) OR
(@Resource[Microsoft.Storage/storageAccounts/blobServices/containers:name] StringEquals 'contosocorp' AND
@Resource[Microsoft.Storage/storageAccounts/blobServices/containers:blobs:path] StringLike
'uploads/contoso/*'))"
$testRa = Get-AzRoleAssignment -Scope $scope -RoleDefinitionName $roleDefinitionName -ObjectId $userObjectID
$testRa.Condition = $condition
$testRa.ConditionVersion = "2.0"
Set-AzRoleAssignment -InputObject $testRa -PassThru

```

Here's how to test this condition.

```

$grantedContainer = "contosocorp"
$localSrcFile = <pathToLocalFile>
$bearerCtx = New-AzStorageContext -StorageAccountName $storageAccountName
Try to set ungranted blob
$content = Set-AzStorageBlobContent -Container $grantedContainer -Blob "Example7.txt" -Context $bearerCtx -
File $localSrcFile
Try to set granted blob
$content = Set-AzStorageBlobContent -Container $grantedContainer -Blob "uploads/contoso/Example7.txt" -
Context $bearerCtx -File $localSrcFile

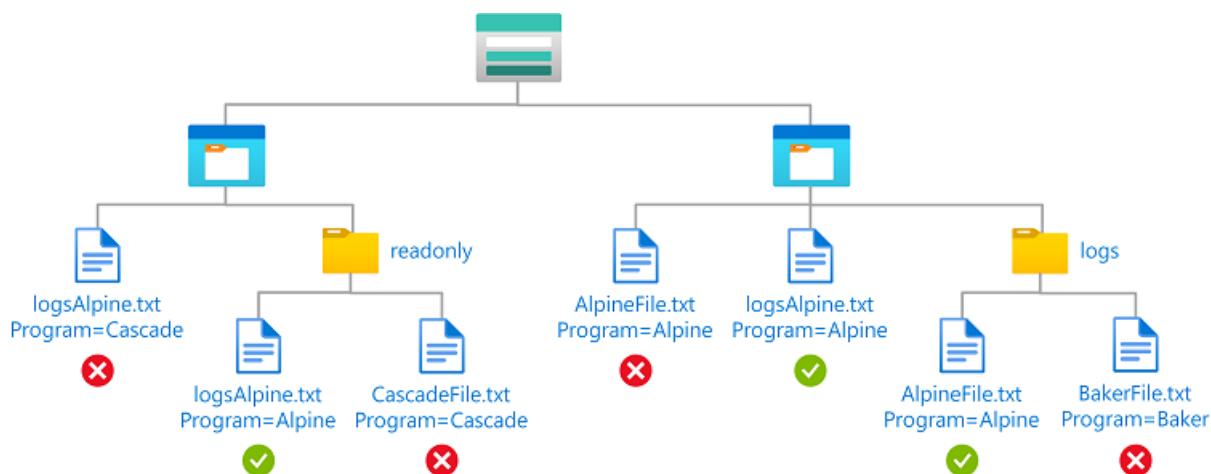
```

### Example: Read blobs with a blob index tag and a path

This condition allows a user to read blobs with a [blob index tag](#) key of Program, a value of Alpine, and a blob path of logs\*. The blob path of logs\* also includes the blob name.

You must add this condition to any role assignments that include the following action.

ACTION	NOTES
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read	
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read	Add this role definition to your role assignment if you want to allow users to read blobs with a blob index tag key of Program, a value of Alpine, and a blob path of logs*. The blob path of logs* also includes the blob name.



```

(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'} AND NOT
SubOperationMatches{'Blob.List'})
)
OR
(
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Program<$key_case_sensitive$>
] StringEquals 'Alpine'
)
)
AND
(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'} AND NOT
SubOperationMatches{'Blob.List'})
)
OR
(
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:path] StringLike 'logs*'
)
)
)

```

#### Azure portal

Here are the settings to add this condition using the Azure portal.

CONDITION #1	SETTING
Actions	<a href="#">Read a blob</a>
Attribute source	Resource
Attribute	<a href="#">Blob index tags [Values in key]</a>
Key	{keyName}
Operator	<a href="#">StringEquals</a>
Value	{keyValue}

## Add role assignment condition

Preview

A condition is an additional check that you can optionally add to your role assignment to provide more fine-grained access control. For example, you can add a condition that requires an object to have a specific tag to read the object. [Learn more](#)

Role Storage Blob Data Reader

Editor type  Visual  Code

**Condition #1** 

To use principal (user) attributes, you must have all of the following: Azure AD Premium P1 or P2 license, Azure AD permissions (such as the Attribute Assignment Administrator), and custom security attributes defined in Azure AD. [Learn more](#)

### 1. Add action \*

Click Add action to select the actions you want to allow if the condition is true. [Learn more](#)

+ Add action 

Action Type	Action	Operation
Data Action	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read	<b>Read a blob</b> 

Select actions

### 2. Build expression

Build one or more expressions. If the expressions evaluate to true, access is allowed to the selected actions. [Learn more](#)

+ Add expression   

1 @Resource[Microsoft.Storage.../tags:Program<\$key\_case\_sensitive\$>] StringEquals 'Alpine' 

Attribute source 	Operator 	Value 
Resource	StringEquals	Alpine
Attribute 	Value <input checked="" type="radio"/> Attribute	
Blob index tags [Values in key]		
Key 	Program	
<input type="checkbox"/> Negate this expression		

+ Add expression

+ Add condition

**Save** **Discard**

CONDITION #2	SETTING
Actions	Read a blob
Attribute source	Resource
Attribute	Blob path
Operator	StringLike
Value	{pathString}

Home >

## Add role assignment condition

**Condition #2**

**1. Add action \***  
Click Add action to select the actions you want to allow if the condition is true. [Learn more](#)

Action Type	Action	Operation
Data Action	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read	Read a blob <a href="#">(i)</a>

**Select actions**

**2. Build expression \***  
Build one or more expressions for the action you selected. If the expressions evaluate to true, access is allowed to the selected action. [Learn more](#)

1 @Resource[Microsoft.Storage/.../blobs:path] StringLike 'logs\*' [\(i\)](#)

Attribute source <a href="#">(i)</a> Resource	Operator <a href="#">(i)</a> StringLike	Value <a href="#">(i)</a> logs*
Attribute <a href="#">(i)</a> Blob path	<input checked="" type="radio"/> Value <input type="radio"/> Attribute	
<input type="checkbox"/> Negate this expression		

**Save** **Discard**

### Azure PowerShell

Here's how to add this condition using Azure PowerShell.

```
$condition = "((!(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'}) AND NOT SubOperationMatches{'Blob.List'})) OR (@Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Program`$key_case_sensitive`$>] StringEquals 'Alpine')) AND ((!(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'}) AND NOT SubOperationMatches{'Blob.List'})) OR (@Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:path] StringLike 'logs*'))"
$testRa = Get-AzRoleAssignment -Scope $scope -RoleDefinitionName $roleDefinitionName -ObjectId $userObjectID
$testRa.Condition = $condition
$testRa.ConditionVersion = "2.0"
Set-AzRoleAssignment -InputObject $testRa -PassThru
```

Here's how to test this condition.

```
$grantedContainer = "contosocorp"
Get new context for request
$bearerCtx = New-AzStorageContext -StorageAccountName $storageAccountName
Try to get ungranted blobs
Wrong name but right tags
$content = Get-AzStorageBlobContent -Container $grantedContainer -Blob "AlpineFile.txt" -Context $bearerCtx
Right name but wrong tags
$content = Get-AzStorageBlobContent -Container $grantedContainer -Blob "logsAlpine.txt" -Context $bearerCtx
Try to get granted blob
$content = Get-AzStorageBlobContent -Container $grantedContainer -Blob "logs/AlpineFile.txt" -Context $bearerCtx
```

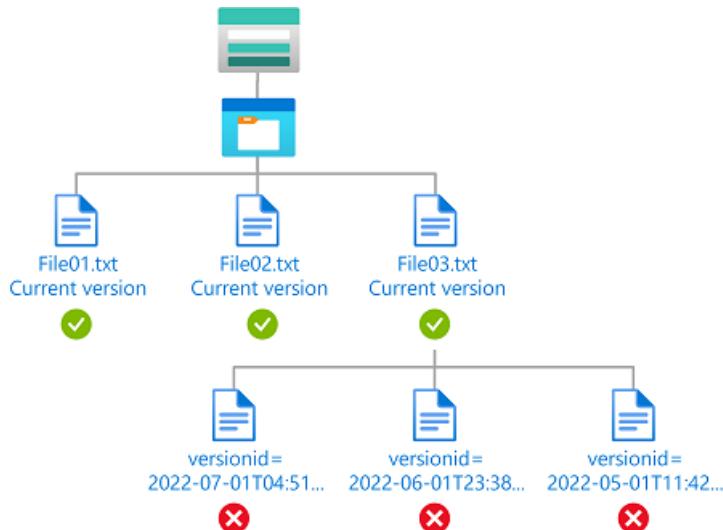
## Blob versions or blob snapshots

### Example: Read only current blob versions

This condition allows a user to only read current blob versions. The user cannot read other blob versions.

You must add this condition to any role assignments that include the following actions.

ACTION	NOTES
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read	
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/runAsSuperUser/action	Add if role definition includes this action, such as Storage Blob Data Owner.



#### Storage Blob Data Owner

```
(
()
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'}) AND NOT
 SubOperationMatches{'Blob.List'}
 AND
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/runAsSuperUser/action'})
)
OR
(
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:isCurrentVersion] BoolEquals
 true
)
)
```

#### Storage Blob Data Reader, Storage Blob Data Contributor

```
(
()
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'}) AND NOT
 SubOperationMatches{'Blob.List'}
)
OR
(
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:isCurrentVersion] BoolEquals
 true
)
)
```

#### Azure portal

Here are the settings to add this condition using the Azure portal.

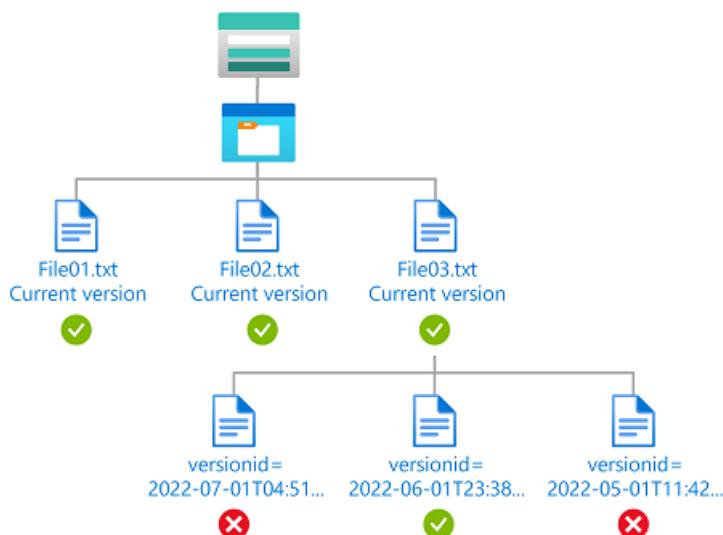
CONDITION #1	SETTING
Actions	Read a blob All data operations for accounts with hierarchical namespace enabled (if applicable)
Attribute source	Resource
Attribute	Is Current Version
Operator	BoolEquals
Value	True

### Example: Read current blob versions and a specific blob version

This condition allows a user to read current blob versions as well as read blobs with a version ID of 2022-06-01T23:38:32.8883645Z. The user cannot read other blob versions. The [Version ID](#) attribute is available only for storage accounts where hierarchical namespace is not enabled.

You must add this condition to any role assignments that include the following action.

ACTION	NOTES
Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read	



```

(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'}) AND NOT
SubOperationMatches{'Blob.List'})
)
OR
(
 @Request[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:versionId] DateTimeEquals '2022-
06-01T23:38:32.8883645Z'
 OR
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:isCurrentVersion] BoolEquals
true
)
)

```

Here are the settings to add this condition using the Azure portal.

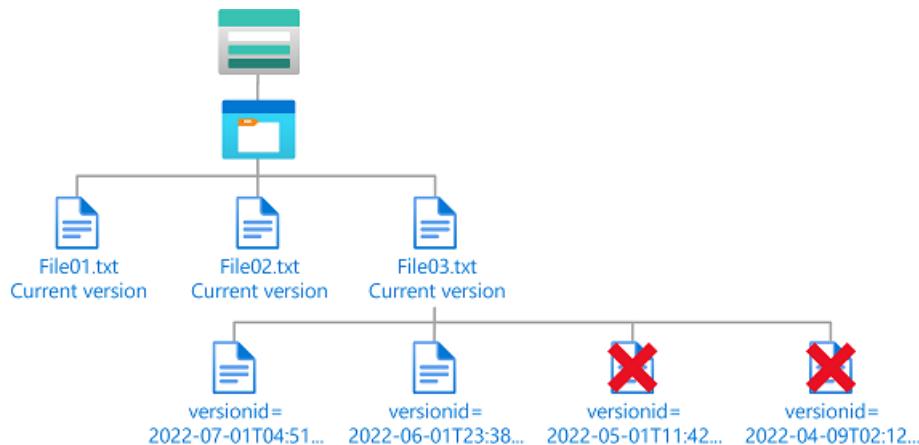
CONDITION #1	SETTING
Actions	Read a blob
Attribute source	Request
Attribute	Version ID
Operator	DateTimeEquals
Value	<blobVersionId>
<b>Expression 2</b>	
Operator	Or
Attribute source	Resource
Attribute	Is Current Version
Operator	BoolEquals
Value	True

#### Example: Delete old blob versions

This condition allows a user to delete versions of a blob that are older than 06/01/2022 to perform clean up. The [Version ID](#) attribute is available only for storage accounts where hierarchical namespace is not enabled.

You must add this condition to any role assignments that include the following actions.

ACTION	NOTES
<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/delete</code>	
<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/deleteBlobVersion/action</code>	



```

(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/delete'})
 AND
 !
 (ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/deleteBlobVersion/action'})
)
OR
(
 @Request[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:versionId] DateTimeLessThan
 '2022-06-01T00:00:00.0Z'
)
)

```

### Azure portal

Here are the settings to add this condition using the Azure portal.

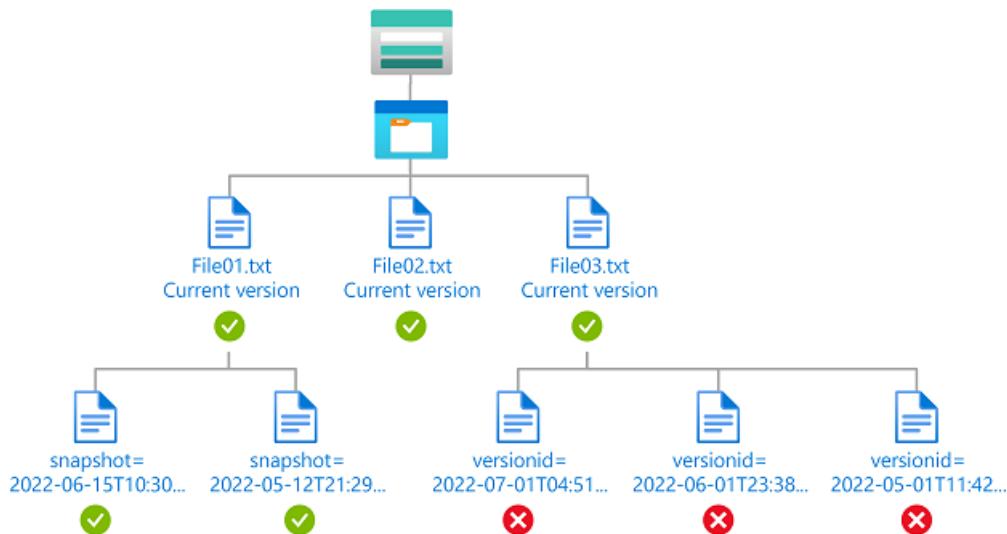
CONDITION #1	SETTING
Actions	<a href="#">Delete a blob</a> <a href="#">Delete a version of a blob</a>
Attribute source	Request
Attribute	<a href="#">Version ID</a>
Operator	<a href="#">DateTimeLessThan</a>
Value	<blobVersionId>

### Example: Read current blob versions and any blob snapshots

This condition allows a user to read current blob versions and any blob snapshots. The [Version ID](#) attribute is available only for storage accounts where hierarchical namespace is not enabled. The [Snapshot](#) attribute is available for storage accounts where hierarchical namespace is not enabled and currently in preview for storage accounts where hierarchical namespace is enabled.

You must add this condition to any role assignments that include the following action.

ACTION	NOTES
<a href="#">Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read</a>	
<a href="#">Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read</a>	Add a role definition that includes this action, such as Storage Blob Data Owner.



### Storage Blob Data Owner

```

(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'} AND NOT
SubOperationMatches{'Blob.List'})
 AND
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/runAsSuperUser/action'})
)
OR
(
 Exists @Request[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:snapshot]
 OR
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:isCurrentVersion] BoolEquals
true
)
)

```

### Storage Blob Data Reader, Storage Blob Data Contributor

```

(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'} AND NOT
SubOperationMatches{'Blob.List'})
)
OR
(
 Exists @Request[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:snapshot]
 OR
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs:isCurrentVersion] BoolEquals
true
)
)

```

### Azure portal

Here are the settings to add this condition using the Azure portal.

CONDITION #1	SETTING
Actions	<a href="#">Read a blob</a> <a href="#">All data operations for accounts with hierarchical namespace enabled (if applicable)</a>
Attribute source	Request

CONDITION #1	SETTING
Attribute	Snapshot
Exists	Checked
Expression 2	
Operator	Or
Attribute source	Resource
Attribute	Is Current Version
Operator	BoolEquals
Value	True

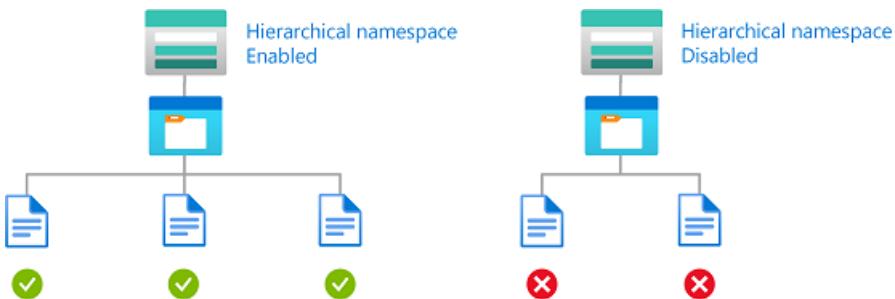
## Hierarchical namespace

### Example: Read only storage accounts with hierarchical namespace enabled

This condition allows a user to only read blobs in storage accounts with [hierarchical namespace](#) enabled. This condition is applicable only at resource group scope or above.

You must add this condition to any role assignments that include the following actions.

ACTION	NOTES
<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read</code>	
<code>Microsoft.Storage/storageAccounts/blobServices/containers/read</code>	Add if role definition includes this action, such as Storage Blob Data Owner.



Storage Blob Data Owner

```

(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'} AND NOT
SubOperationMatches{'Blob.List'})
 AND
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/runAsSuperUser/action'})
)
OR
(
 @Resource[Microsoft.Storage/storageAccounts:isHnsEnabled] BoolEquals true
)
)

```

## Storage Blob Data Reader, Storage Blob Data Contributor

```

(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'} AND NOT
SubOperationMatches{'Blob.List'})
)
OR
(
 @Resource[Microsoft.Storage/storageAccounts:isHnsEnabled] BoolEquals true
)
)

```

### Azure portal

Here are the settings to add this condition using the Azure portal.

CONDITION #1	SETTING
Actions	Read a blob All data operations for accounts with hierarchical namespace enabled (if applicable)
Attribute source	Resource
Attribute	Is hierarchical namespace enabled
Operator	BoolEquals
Value	True

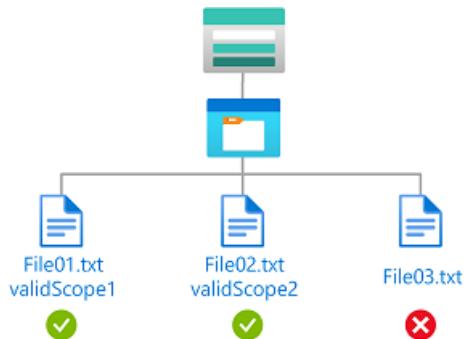
## Encryption scope

### Example: Read blobs with specific encryption scopes

This condition allows a user to read blobs encrypted with encryption scope `validScope1` or `validScope2`.

You must add this condition to any role assignments that include the following action.

ACTION	NOTES
<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read</code>	
<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/runAsSuperUser/action</code>	Add this definition if your role definition includes this action, such as Storage Blob Data Owner.



```

(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'} AND NOT
SubOperationMatches{'Blob.List'})
)
OR
(
 @Resource[Microsoft.Storage/storageAccounts/encryptionScopes:name] ForAnyOfAnyValues:StringEquals
 {'validScope1', 'validScope2'}
)
)

```

#### Azure portal

Here are the settings to add this condition using the Azure portal.

CONDITION #1	SETTING
Actions	Read a blob
Attribute source	Resource
Attribute	Encryption scope name
Operator	ForAnyOfAnyValues:StringEquals
Value	<scopeName>

#### Example: Read or write blobs in named storage account with specific encryption scope

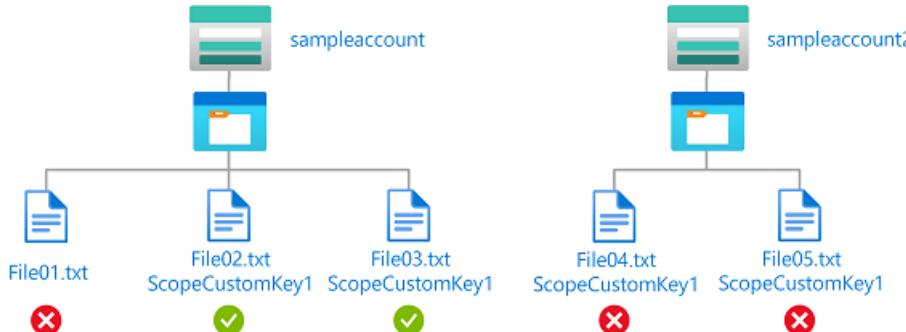
This condition allows a user to read or write blobs in a storage account named `sampleaccount` and encrypted with encryption scope `ScopeCustomKey1`. If blobs are not encrypted or decrypted with `ScopeCustomKey1`, request will return forbidden.

You must add this condition to any role assignments that include the following actions.

ACTION	NOTES
<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read</code>	
<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write</code>	
<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action</code>	
<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/delete</code>	Add this definition if your role definition includes this action, such as Storage Blob Data Owner.

## NOTE

Since encryption scopes for different storage accounts could be different, it's recommended to use the `storageAccounts:name` attribute with the `encryptionScopes:name` attribute to restrict the specific encryption scope to be allowed.



```
(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'} AND NOT
SubOperationMatches{'Blob.List'})
 AND
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write'})
 AND
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action'})
)
OR
(
 @Resource[Microsoft.Storage/storageAccounts:name] StringEquals 'sampleaccount'
 AND
 @Resource[Microsoft.Storage/storageAccounts/encryptionScopes:name] ForAnyOfAnyValues:StringEquals
{'ScopeCustomKey1'}
)
)
```

## Azure portal

Here are the settings to add this condition using the Azure portal.

CONDITION #1	SETTING
Actions	Read a blob Write to a blob Create a blob or snapshot, or append data
Attribute source	Resource
Attribute	Account name
Operator	StringEquals
Value	<accountName>
Expression 2	
Operator	And
Attribute source	Resource

CONDITION #1	SETTING
Attribute	Encryption scope name
Operator	ForAnyOfAnyValues:StringEquals
Value	<scopeName>

## Principal attributes

### Example: Read or write blobs based on blob index tags and custom security attributes

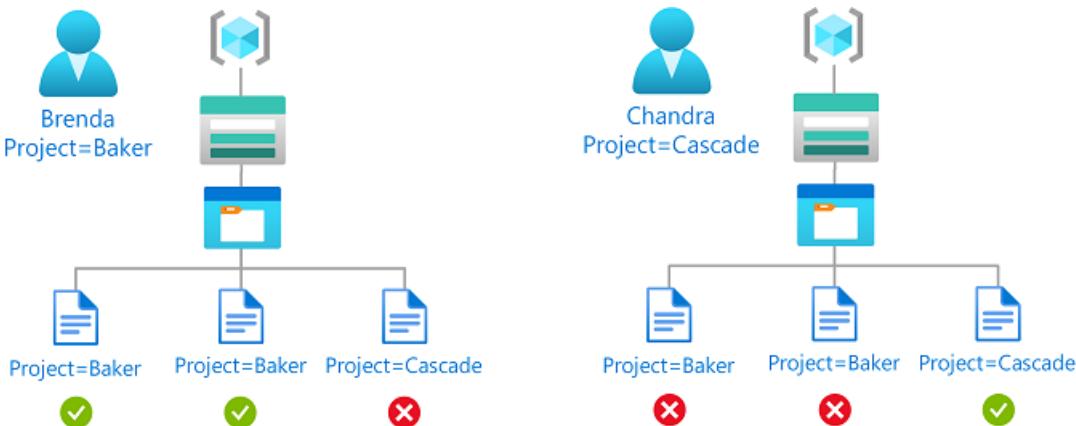
This condition allows read or write access to blobs if the user has a [custom security attribute](#) that matches the [blob index tag](#).

For example, if Brenda has the attribute `Project=Baker`, she can only read or write blobs with the `Project=Baker` blob index tag. Similarly, Chandra can only read or write blobs with `Project=Cascade`.

You must add this condition to any role assignments that include the following actions.

ACTION	NOTES
<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read</code>	
<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write</code>	
<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action</code>	
<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/delete</code>	Add this action to your role definition if you include this action, such as Storage Blob Data Owner.

For more information, see [Allow read access to blobs based on tags and custom security attributes](#).



```

(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'} AND NOT
SubOperationMatches{'Blob.List'})
)
OR
(
 @Principal[Microsoft.Directory/CustomSecurityAttributes/Id:Engineering_Project] StringEquals
@Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Project<$key_case_sensitive$>
]
)
)
AND
(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/write'} AND
SubOperationMatches{'Blob.WriteWithTagHeaders'})
AND
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/add/action'} AND
SubOperationMatches{'Blob.WriteWithTagHeaders'})
)
OR
(
 @Principal[Microsoft.Directory/CustomSecurityAttributes/Id:Engineering_Project] StringEquals
@Request[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Project<$key_case_sensitive$>]
)
)
)

```

## Azure portal

Here are the settings to add this condition using the Azure portal.

CONDITION #1	SETTING
Actions	<a href="#">Read a blob conditions</a>
Attribute source	<a href="#">Principal</a>
Attribute	<attributeset>_<key>
Operator	<a href="#">StringEquals</a>
Option	Attribute
Attribute source	Resource
Attribute	<a href="#">Blob index tags [Values in key]</a>
Key	<key>
CONDITION #2	SETTING
Actions	<a href="#">Write to a blob with blob index tags</a> <a href="#">Write to a blob with blob index tags</a>
Attribute source	<a href="#">Principal</a>
Attribute	<attributeset>_<key>

CONDITION #2	SETTING
Operator	StringEquals
Option	Attribute
Attribute source	Request
Attribute	Blob index tags [Values in key]
Key	<key>

### Example: Read blobs based on blob index tags and multi-value custom security attributes

This condition allows read access to blobs if the user has a [custom security attribute](#) with any values that matches the [blob index tag](#).

For example, if Chandra has the Project attribute with the values Baker and Cascade, she can only read blobs with the `Project=Baker` or `Project=Cascade` blob index tag.

You must add this condition to any role assignments that include the following action.

ACTION	NOTES
<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read</code>	
<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read</code>	Add this definition to your role definition if you want to allow users to read blobs. Additional definitions include this action, such as Storage Blob Data Owner.

For more information, see [Allow read access to blobs based on tags and custom security attributes](#).



```

(
(
 !(ActionMatches{'Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read'} AND NOT
SubOperationMatches{'Blob.List'})
)
OR
(
 @Resource[Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags:Project<$key_case_sensitive$>
] ForAnyOfAnyValues:StringEquals
@Principal[Microsoft.Directory/CustomSecurityAttributes/Id:Engineering_Project]
)
)

```

## Azure portal

Here are the settings to add this condition using the Azure portal.

CONDITION #1	SETTING
Actions	Read a blob conditions
Attribute source	Resource
Attribute	Blob index tags [Values in key]
Key	<key>
Operator	ForAnyOfAnyValues:StringEquals
Option	Attribute
Attribute source	Principal
Attribute	<attributeset>_<key>

## Next steps

- [Tutorial: Add a role assignment condition to restrict access to blobs using the Azure portal \(preview\)](#)
- [Actions and attributes for Azure role assignment conditions in Azure Storage \(preview\)](#)
- [Azure role assignment condition format and syntax \(preview\)](#)
- [Troubleshoot Azure role assignment conditions \(preview\)](#)

# Grant limited access to Azure Storage resources using shared access signatures (SAS)

8/22/2022 • 13 minutes to read • [Edit Online](#)

A shared access signature (SAS) provides secure delegated access to resources in your storage account. With a SAS, you have granular control over how a client can access your data. For example:

- What resources the client may access.
- What permissions they have to those resources.
- How long the SAS is valid.

## Types of shared access signatures

Azure Storage supports three types of shared access signatures:

- User delegation SAS
- Service SAS
- Account SAS

### 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 \(REST API\)](#).

### 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 \(REST API\)](#).

### 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.

You can also delegate access to the following:

- Service-level operations (For example, the [Get/Set Service Properties](#) and [Get Service Stats](#) operations).
- Read, write, and delete operations that aren't permitted with a service SAS.

For more information about the account SAS, [Create an account SAS \(REST API\)](#).

#### **NOTE**

Microsoft recommends that you use Azure AD credentials when possible as a security best practice, rather than using the account key, which can be more easily compromised. When your application design requires shared access signatures for access to Blob storage, use Azure AD credentials to create a user delegation SAS when possible for superior security. For more information, see [Authorize access to data in Azure Storage](#).

A shared access signature can take one of the following two forms:

- **Ad hoc SAS.** When you create an ad hoc SAS, the start time, expiry time, and permissions are specified in the SAS URI. Any type of SAS can be an ad hoc SAS.
- **Service SAS with stored access policy.** A stored access policy is defined on a resource container, which can be a blob container, table, queue, or file share. The stored access policy can be used to manage constraints for one or more service shared access signatures. When you associate a service SAS with a stored access policy, the SAS inherits the constraints—the start time, expiry time, and permissions—defined for the stored access policy.

#### **NOTE**

A user delegation SAS or an account SAS must be an ad hoc SAS. Stored access policies are not supported for the user delegation SAS or the account SAS.

## How a shared access signature works

A shared access signature is a signed URI that points to one or more storage resources. The URI includes a token that contains a special set of query parameters. The token indicates how the resources may be accessed by the client. One of the query parameters, the signature, is constructed from the SAS parameters and signed with the key that was used to create the SAS. This signature is used by Azure Storage to authorize access to the storage resource.

#### **NOTE**

It's not possible to audit the generation of SAS tokens. Any user that has privileges to generate a SAS token, either by using the account key, or via an Azure role assignment, can do so without the knowledge of the owner of the storage account. Be careful to restrict permissions that allow users to generate SAS tokens. To prevent users from generating a SAS that is signed with the account key for blob and queue workloads, you can disallow Shared Key access to the storage account. For more information, see [Prevent authorization with Shared Key](#).

## SAS signature and authorization

You can sign a SAS token with a user delegation key or with a storage account key (Shared Key).

### **Signing a SAS token with a user delegation key**

You can sign a SAS token by using a *user delegation key* that was created using Azure Active Directory (Azure AD) credentials. A user delegation SAS is signed with the user delegation key.

To get the key, and then create the SAS, an Azure AD security principal must be assigned an Azure role that includes the `Microsoft.Storage/storageAccounts/blobServices/generateUserDelegationKey` action. For more information, see [Create a user delegation SAS \(REST API\)](#).

### **Signing a SAS token with an account key**

Both a service SAS and an account SAS are signed with the storage account key. To create a SAS that is signed with the account key, an application must have access to the account key.

When a request includes a SAS token, that request is authorized based on how that SAS token is signed. The access key or credentials that you use to create a SAS token are also used by Azure Storage to grant access to a client that possesses the SAS.

The following table summarizes how each type of SAS token is authorized.

TYPE OF SAS	TYPE OF AUTHORIZATION
User delegation SAS (Blob storage only)	Azure AD
Service SAS	Shared Key
Account SAS	Shared Key

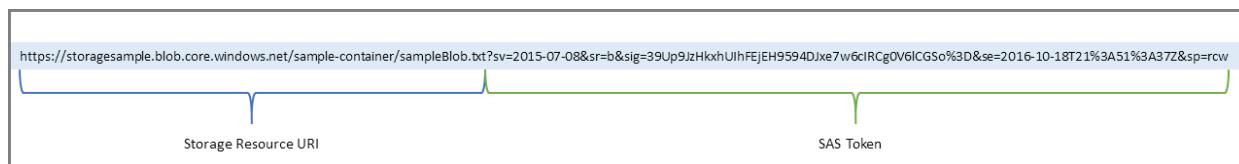
Microsoft recommends using a user delegation SAS when possible for superior security.

### SAS token

The SAS token is a string that you generate on the client side, for example by using one of the Azure Storage client libraries. The SAS token is not tracked by Azure Storage in any way. You can create an unlimited number of SAS tokens on the client side. After you create a SAS, you can distribute it to client applications that require access to resources in your storage account.

Client applications provide the SAS URI to Azure Storage as part of a request. Then, the service checks the SAS parameters and the signature to verify that it is valid. If the service verifies that the signature is valid, then the request is authorized. Otherwise, the request is declined with error code 403 (Forbidden).

Here's an example of a service SAS URI, showing the resource URI and the SAS token. Because the SAS token comprises the URI query string, the resource URI must be followed first by a question mark, and then by the SAS token:

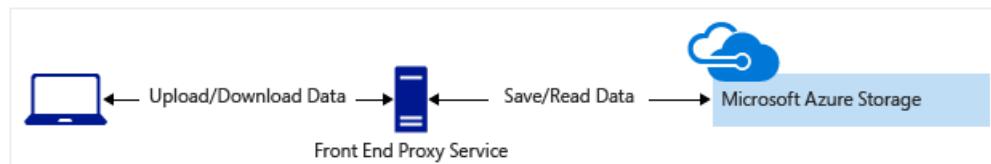


## When to use a shared access signature

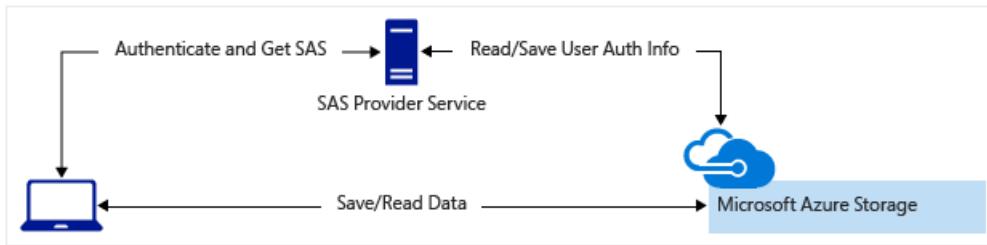
Use a SAS to give secure access to resources in your storage account to any client who does not otherwise have permissions to those resources.

A common scenario where a SAS is useful is a service where users read and write their own data to your storage account. In a scenario where a storage account stores user data, there are two typical design patterns:

1. Clients upload and download data via a front-end proxy service, which performs authentication. This front-end proxy service allows the validation of business rules. But for large amounts of data, or high-volume transactions, creating a service that can scale to match demand may be expensive or difficult.



2. A lightweight service authenticates the client as needed and then generates a SAS. Once the client application receives the SAS, it can access storage account resources directly. Access permissions are defined by the SAS and for the interval allowed by the SAS. The SAS mitigates the need for routing all data through the front-end proxy service.



Many real-world services may use a hybrid of these two approaches. For example, some data might be processed and validated via the front-end proxy. Other data is saved and/or read directly using SAS.

Additionally, a SAS is required to authorize access to the source object in a copy operation in certain scenarios:

- When you copy a blob to another blob that resides in a different storage account.  
You can optionally use a SAS to authorize access to the destination blob as well.
- When you copy a file to another file that resides in a different storage account.  
You can optionally use a SAS to authorize access to the destination file as well.
- When you copy a blob to a file, or a file to a blob.  
You must use a SAS even if the source and destination objects reside within the same storage account.

## Best practices when using SAS

When you use shared access signatures in your applications, you need to be aware of two potential risks:

- If a SAS is leaked, it can be used by anyone who obtains it, which can potentially compromise your storage account.
- If a SAS provided to a client application expires and the application is unable to retrieve a new SAS from your service, then the application's functionality may be hindered.

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. Then, they can use that SAS just as the intended user could have. This can potentially compromise 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.
- **Configure a SAS expiration policy for the storage account.** A SAS expiration policy specifies a recommended interval over which the SAS is valid. SAS expiration policies apply to a service SAS or an account SAS. When a user generates service SAS or an account SAS with a validity interval that is larger than the recommended interval, they'll see a warning. If Azure Storage logging with Azure Monitor is enabled, then an entry is written to the Azure Storage logs. To learn more, see [Create an expiration policy for shared access signatures](#).
- **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. This might be unnecessary in some cases. For example, you might intend for the SAS to be used for a small number of immediate, short-lived operations. These operations are expected to be completed within the expiration period. As a result, you are not expecting the SAS to be renewed. However, if you have a client that is routinely making requests via SAS, then the possibility of expiration comes into play.
- **Be careful with SAS start time.** If you set the start time for a SAS to the current time, failures might occur intermittently for the first few minutes. This is due to different machines having slightly different current times (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. Any policies that specify a longer term than 1 hour will fail.
- **Be careful with SAS datetime format.** For some utilities (such as AzCopy), date/time values must be formatted as '+%Y-%m-%dT%H:%M:%SZ'. This format specifically includes the seconds.
- **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 you plan to validate data, perform that 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.** Authorization failures can occur because of an outage in your SAS provider service. They can also occur from an inadvertent removal of a stored access policy. You can use Azure Monitor and storage analytics logging to observe any spike in these types of authorization failures. For more information, see [Azure Storage metrics in Azure Monitor](#) and [Azure Storage Analytics logging](#).

#### **NOTE**

Storage doesn't track the number of shared access signatures that have been generated for a storage account, and no API can provide this detail. If you need to know the number of shared access signatures that have been generated for a storage account, you must track the number manually.

## Get started with SAS

To get started with shared access signatures, see the following articles for each SAS type.

### **User delegation SAS**

- [Create a user delegation SAS for a container or blob with PowerShell](#)
- [Create a user delegation SAS for a container or blob with the Azure CLI](#)
- [Create a user delegation SAS for a container or blob with .NET](#)

### **Service SAS**

- [Create a service SAS for a container or blob with .NET](#)

### **Account SAS**

- [Create an account SAS with .NET](#)

## Next steps

- [Delegate access with a shared access signature \(REST API\)](#)
- [Create a user delegation SAS \(REST API\)](#)
- [Create a service SAS \(REST API\)](#)
- [Create an account SAS \(REST API\)](#)

# Use the Azure Storage resource provider to access management resources

8/22/2022 • 4 minutes to read • [Edit Online](#)

Azure Resource Manager is the deployment and management service for Azure. The Azure Storage resource provider is a service that is based on Azure Resource Manager and that provides access to management resources for Azure Storage. You can use the Azure Storage resource provider to create, update, manage, and delete resources such as storage accounts, private endpoints, and account access keys. For more information about Azure Resource Manager, see [Azure Resource Manager overview](#).

You can use the Azure Storage resource provider to perform actions such as creating or deleting a storage account or getting a list of storage accounts in a subscription. To authorize requests against the Azure Storage resource provider, use Azure Active Directory (Azure AD). This article describes how to assign permissions to management resources, and points to examples that show how to make requests against the Azure Storage resource provider.

## Management resources versus data resources

Microsoft provides two REST APIs for working with Azure Storage resources. These APIs form the basis of all actions you can perform against Azure Storage. The Azure Storage REST API enables you to work with data in your storage account, including blob, queue, file, and table data. The Azure Storage resource provider REST API enables you to work with the storage account and related resources.

A request that reads or writes blob data requires different permissions than a request that performs a management operation. Azure RBAC provides fine-grained control over permissions to both types of resources. When you assign an Azure role to a security principal, make sure that you understand what permissions that principal will be granted. For a detailed reference that describes which actions are associated with each Azure built-in role, see [Azure built-in roles](#).

Azure Storage supports using Azure AD to authorize requests against Blob and Queue storage. For information about Azure roles for blob and queue data operations, see [Authorize access to blobs and queues using Active Directory](#).

## Assign management permissions with Azure role-based access control (Azure RBAC)

Every Azure subscription has an associated Azure Active Directory that manages users, groups, and applications. A user, group, or application is also referred to as a security principal in the context of the [Microsoft identity platform](#). You can grant access to resources in a subscription to a security principal that is defined in the Active Directory by using Azure role-based access control (Azure RBAC).

When you assign an Azure role to a security principal, you also indicate the scope at which the permissions granted by the role are in effect. For management operations, you can assign a role at the level of the subscription, the resource group, or the storage account. You can assign an Azure role to a security principal by using the [Azure portal](#), the [Azure classic CLI](#), [PowerShell](#), or the [Azure Storage resource provider REST API](#).

For more information, see [What is Azure role-based access control \(Azure RBAC\)?](#) and [Classic subscription administrator roles, Azure roles, and Azure AD administrator roles](#).

### Built-in roles for management operations

Azure provides built-in roles that grant permissions to call management operations. Azure Storage also provides built-in roles specifically for use with the Azure Storage resource provider.

Built-in roles that grant permissions to call storage management operations include the roles described in the following table:

AZURE ROLE	DESCRIPTION	INCLUDES ACCESS TO ACCOUNT KEYS?
Owner	Can manage all storage resources and access to resources.	Yes, provides permissions to view and regenerate the storage account keys.
Contributor	Can manage all storage resources, but cannot manage access to resources.	Yes, provides permissions to view and regenerate the storage account keys.
Reader	Can view information about the storage account, but cannot view the account keys.	No.
Storage Account Contributor	Can manage the storage account, get information about the subscription's resource groups and resources, and create and manage subscription resource group deployments.	Yes, provides permissions to view and regenerate the storage account keys.
User Access Administrator	Can manage access to the storage account.	Yes, permits a security principal to assign any permissions to themselves and others.
Virtual Machine Contributor	Can manage virtual machines, but not the storage account to which they are connected.	Yes, provides permissions to view and regenerate the storage account keys.

The third column in the table indicates whether the built-in role supports the [Microsoft.Storage/storageAccounts/listkeys/action](#). This action grants permissions to read and regenerate the storage account keys. Permissions to access Azure Storage management resources do not also include permissions to access data. However, if a user has access to the account keys, then they can use the account keys to access Azure Storage data via Shared Key authorization.

### Custom roles for management operations

Azure also supports defining Azure custom roles for access to management resources. For more information about custom roles, see [Azure custom roles](#).

## Code samples

For code examples that show how to authorize and call management operations from the Azure Storage management libraries, see the following samples:

- [.NET](#)
- [Java](#)
- [Node.js](#)
- [Python](#)

## Azure Resource Manager versus classic deployments

The Resource Manager and classic deployment models represent two different ways of deploying and managing

your Azure solutions. Microsoft recommends using the Azure Resource Manager deployment model when you create a new storage account. If possible, Microsoft also recommends that you recreate existing classic storage accounts with the Resource Manager model. Although you can create a storage account using the classic deployment model, the classic model is less flexible and will eventually be deprecated.

For more information about Azure deployment models, see [Resource Manager and classic deployment](#).

## Next steps

- [Azure Resource Manager overview](#)
- [What is Azure role-based access control \(Azure RBAC\)?](#)
- [Scalability targets for the Azure Storage resource provider](#)

# Security recommendations for Blob storage

8/22/2022 • 9 minutes to read • [Edit Online](#)

This article contains security recommendations for Blob storage. Implementing these recommendations will help you fulfill your security obligations as described in our shared responsibility model. For more information on how Microsoft fulfills service provider responsibilities, see [Shared responsibility in the cloud](#).

Some of the recommendations included in this article can be automatically monitored by Microsoft Defender for Cloud, which is the first line of defense in protecting your resources in Azure. For information on Microsoft Defender for Cloud, see [What is Microsoft Defender for Cloud?](#)

Microsoft Defender for Cloud periodically analyzes the security state of your Azure resources to identify potential security vulnerabilities. It then provides you with recommendations on how to address them. For more information on Microsoft Defender for Cloud recommendations, see [Review your security recommendations](#).

## Data protection

RECOMMENDATION	COMMENTS	DEFENDER FOR CLOUD
Use the Azure Resource Manager deployment model	Create new storage accounts using the Azure Resource Manager deployment model for important security enhancements, including superior Azure role-based access control (Azure RBAC) and auditing, Resource Manager-based deployment and governance, access to managed identities, access to Azure Key Vault for secrets, and Azure AD-based authentication and authorization for access to Azure Storage data and resources. If possible, migrate existing storage accounts that use the classic deployment model to use Azure Resource Manager. For more information about Azure Resource Manager, see <a href="#">Azure Resource Manager overview</a> .	-
Enable Microsoft Defender for all of your storage accounts	Microsoft Defender for Storage provides an additional layer of security intelligence that detects unusual and potentially harmful attempts to access or exploit storage accounts. Security alerts are triggered in Microsoft Defender for Cloud when anomalies in activity occur and are also sent via email to subscription administrators, with details of suspicious activity and recommendations on how to investigate and remediate threats. For more information, see <a href="#">Configure Microsoft Defender for Storage</a> .	Yes

RECOMMENDATION	COMMENTS	DEFENDER FOR CLOUD
Turn on soft delete for blobs	Soft delete for blobs enables you to recover blob data after it has been deleted. For more information on soft delete for blobs, see <a href="#">Soft delete for Azure Storage blobs</a> .	-
Turn on soft delete for containers	Soft delete for containers enables you to recover a container after it has been deleted. For more information on soft delete for containers, see <a href="#">Soft delete for containers</a> .	-
Lock storage account to prevent accidental or malicious deletion or configuration changes	Apply an Azure Resource Manager lock to your storage account to protect the account from accidental or malicious deletion or configuration change. Locking a storage account does not prevent data within that account from being deleted. It only prevents the account itself from being deleted. For more information, see <a href="#">Apply an Azure Resource Manager lock to a storage account</a> .	-
Store business-critical data in immutable blobs	Configure legal holds and time-based retention policies to store blob data in a WORM (Write Once, Read Many) state. Blobs stored immutably can be read, but cannot be modified or deleted for the duration of the retention interval. For more information, see <a href="#">Store business-critical blob data with immutable storage</a> .	-
Require secure transfer (HTTPS) to the storage account	When you require secure transfer for a storage account, all requests to the storage account must be made over HTTPS. Any requests made over HTTP are rejected. Microsoft recommends that you always require secure transfer for all of your storage accounts. For more information, see <a href="#">Require secure transfer to ensure secure connections</a> .	-
Limit shared access signature (SAS) tokens to HTTPS connections only	Requiring HTTPS when a client uses a SAS token to access blob data helps to minimize the risk of eavesdropping. For more information, see <a href="#">Grant limited access to Azure Storage resources using shared access signatures (SAS)</a> .	-

## Identity and access management

RECOMMENDATION	COMMENTS	DEFENDER FOR CLOUD
Use Azure Active Directory (Azure AD) to authorize access to blob data	Azure AD provides superior security and ease of use over Shared Key for authorizing requests to Blob storage. For more information, see <a href="#">Authorize access to data in Azure Storage</a> .	-
Keep in mind the principal of least privilege when assigning permissions to an Azure AD security principal via Azure RBAC	When assigning a role to a user, group, or application, grant that security principal only those permissions that are necessary for them to perform their tasks. Limiting access to resources helps prevent both unintentional and malicious misuse of your data.	-
Use a user delegation SAS to grant limited access to blob data to clients	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 is analogous to a service SAS in terms of its scope and function, but offers security benefits over the service SAS. For more information, see <a href="#">Grant limited access to Azure Storage resources using shared access signatures (SAS)</a> .	-
Secure your account access keys with Azure Key Vault	Microsoft recommends using Azure AD to authorize requests to Azure Storage. However, if you must use Shared Key authorization, then secure your account keys with Azure Key Vault. You can retrieve the keys from the key vault at runtime, instead of saving them with your application. For more information about Azure Key Vault, see <a href="#">Azure Key Vault overview</a> .	-
Regenerate your account keys periodically	Rotating the account keys periodically reduces the risk of exposing your data to malicious actors.	-
Disallow Shared Key authorization	When you disallow Shared Key authorization for a storage account, Azure Storage rejects all subsequent requests to that account that are authorized with the account access keys. Only secured requests that are authorized with Azure AD will succeed. For more information, see <a href="#">Prevent Shared Key authorization for an Azure Storage account</a> .	-
Keep in mind the principal of least privilege when assigning permissions to a SAS	When creating a SAS, specify only those permissions that are required by the client to perform its function. Limiting access to resources helps prevent both unintentional and malicious misuse of your data.	-

RECOMMENDATION	COMMENTS	DEFENDER FOR CLOUD
Have a revocation plan in place for any SAS that you issue to clients	If a SAS is compromised, you will want to revoke that SAS as soon as possible. To revoke a user delegation SAS, revoke the user delegation key to quickly invalidate all signatures associated with that key. To revoke a service SAS that is associated with a stored access policy, you can delete the stored access policy, rename the policy, or change its expiry time to a time that is in the past. For more information, see <a href="#">Grant limited access to Azure Storage resources using shared access signatures (SAS)</a> .	-
If a service SAS is not associated with a stored access policy, then set the expiry time to one hour or less	A service SAS that is not associated with a stored access policy cannot be revoked. For this reason, limiting the expiry time so that the SAS is valid for one hour or less is recommended.	-
Disable anonymous public read access to containers and blobs	Anonymous public read access to a container and its blobs grants read-only access to those resources to any client. Avoid enabling public read access unless your scenario requires it. To learn how to disable anonymous public access for a storage account, see <a href="#">Configure anonymous public read access for containers and blobs</a> .	-

## Networking

RECOMMENDATION	COMMENTS	DEFENDER FOR CLOUD
Configure the minimum required version of Transport Layer Security (TLS) for a storage account.	Require that clients use a more secure version of TLS to make requests against an Azure Storage account by configuring the minimum version of TLS for that account. For more information, see <a href="#">Configure minimum required version of Transport Layer Security (TLS) for a storage account</a>	-
Enable the <b>Secure transfer required</b> option on all of your storage accounts	When you enable the <b>Secure transfer required</b> option, all requests made against the storage account must take place over secure connections. Any requests made over HTTP will fail. For more information, see <a href="#">Require secure transfer in Azure Storage</a> .	Yes

RECOMMENDATION	COMMENTS	DEFENDER FOR CLOUD
Enable firewall rules	<p>Configure firewall rules to limit access to your storage account to requests that originate from specified IP addresses or ranges, or from a list of subnets in an Azure Virtual Network (VNet). For more information about configuring firewall rules, see <a href="#">Configure Azure Storage firewalls and virtual networks</a>.</p>	-
Allow trusted Microsoft services to access the storage account	<p>Turning on firewall rules for your storage account blocks incoming requests for data by default, unless the requests originate from a service operating within an Azure Virtual Network (VNet) or from allowed public IP addresses. Requests that are blocked include those from other Azure services, from the Azure portal, from logging and metrics services, and so on. You can permit requests from other Azure services by adding an exception to allow trusted Microsoft services to access the storage account. For more information about adding an exception for trusted Microsoft services, see <a href="#">Configure Azure Storage firewalls and virtual networks</a>.</p>	-
Use private endpoints	<p>A private endpoint assigns a private IP address from your Azure Virtual Network (VNet) to the storage account. It secures all traffic between your VNet and the storage account over a private link. For more information about private endpoints, see <a href="#">Connect privately to a storage account using Azure Private Endpoint</a>.</p>	-
Use VNet service tags	<p>A service tag represents a group of IP address prefixes from a given Azure service. Microsoft manages the address prefixes encompassed by the service tag and automatically updates the service tag as addresses change. For more information about service tags supported by Azure Storage, see <a href="#">Azure service tags overview</a>. For a tutorial that shows how to use service tags to create outbound network rules, see <a href="#">Restrict access to PaaS resources</a>.</p>	-
Limit network access to specific networks	<p>Limiting network access to networks hosting clients requiring access reduces the exposure of your resources to network attacks.</p>	Yes

RECOMMENDATION	COMMENTS	DEFENDER FOR CLOUD
Configure network routing preference	You can configure network routing preference for your Azure storage account to specify how network traffic is routed to your account from clients over the Internet using the Microsoft global network or Internet routing. For more information, see <a href="#">Configure network routing preference for Azure Storage</a> .	-

## Logging/Monitoring

RECOMMENDATION	COMMENTS	DEFENDER FOR CLOUD
Track how requests are authorized	Enable Azure Storage logging to track how each request made against Azure Storage was authorized. The logs indicate whether a request was made anonymously, by using an OAuth 2.0 token, by using Shared Key, or by using a shared access signature (SAS). For more information, see <a href="#">Monitoring Azure Blob Storage with Azure Monitor</a> or <a href="#">Azure Storage analytics logging with Classic Monitoring</a> .	-
Set up alerts in Azure Monitor	Configure log alerts to evaluate resources logs at a set frequency and fire an alert based on the results. For more information, see <a href="#">Log alerts in Azure Monitor</a> .	-

## Next steps

- [Azure security documentation](#)
- [Secure development documentation](#).

# Azure Storage encryption for data at rest

8/22/2022 • 7 minutes to read • [Edit Online](#)

Azure Storage uses service-side encryption (SSE) to automatically encrypt your data when it is persisted to the cloud. Azure Storage encryption protects your data and to help you to meet your organizational security and compliance commitments.

Microsoft recommends using service-side encryption to protect your data for most scenarios. However, the Azure Storage client libraries for Blob Storage and Queue Storage also provide client-side encryption for customers who need to encrypt data on the client. For more information, see [Client-side encryption for blobs and queues](#).

## About Azure Storage service-side encryption

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 storage accounts, including both Resource Manager and classic storage accounts. Azure Storage encryption 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.

Data in a storage account is encrypted regardless of performance tier (standard or premium), access tier (hot or cool), or deployment model (Azure Resource Manager or classic). All blobs in the archive tier are also encrypted. All Azure Storage redundancy options support encryption, and all data in both the primary and secondary regions is encrypted when geo-replication is enabled. All Azure Storage resources are encrypted, including blobs, disks, files, queues, and tables. All object metadata is also encrypted. There is no additional cost for Azure Storage encryption.

Every block blob, append blob, or page blob that was written to Azure Storage after October 20, 2017 is encrypted. Blobs created prior to this date continue to be encrypted by a background process. To force the encryption of a blob that was created before October 20, 2017, you can rewrite the blob. To learn how to check the encryption status of a blob, see [Check the encryption status of a blob](#).

For more information about the cryptographic modules underlying Azure Storage encryption, see [Cryptography API: Next Generation](#).

For information about encryption and key management for Azure managed disks, see [Server-side encryption of Azure managed disks](#).

## About encryption key management

Data in a new storage account is encrypted with Microsoft-managed keys by default. You can continue to rely on Microsoft-managed keys for the encryption of your data, 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 use either type of key management, or both:

- You can specify a *customer-managed key* to use for encrypting and decrypting data in Blob Storage and in Azure Files.<sup>1,2</sup> Customer-managed keys must be stored in Azure Key Vault or Azure Key Vault Managed Hardware Security Model (HSM) (preview). For more information about customer-managed keys, see [Use customer-managed keys for Azure Storage encryption](#).
- 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. For more information about customer-provided keys, see [Provide an encryption key on a request to Blob Storage](#).

By default, a storage account is encrypted with a key that is scoped to the entire storage account. Encryption scopes enable you to manage encryption with a key that is scoped to a container or an individual blob. You can use encryption scopes to create secure boundaries between data that resides in the same storage account but belongs to different customers. Encryption scopes can use either Microsoft-managed keys or customer-managed keys. For more information about encryption scopes, see [Encryption scopes for Blob storage](#).

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

KEY MANAGEMENT PARAMETER	MICROSOFT-MANAGED KEYS	CUSTOMER-MANAGED KEYS	CUSTOMER-PROVIDED KEYS
Encryption/decryption operations	Azure	Azure	Azure
Azure Storage services supported	All	Blob Storage, Azure Files <sup>1,2</sup>	Blob Storage
Key storage	Microsoft key store	Azure Key Vault or Key Vault HSM	Customer's own key store
Key rotation responsibility	Microsoft	Customer	Customer
Key control	Microsoft	Customer	Customer
Key scope	Account (default), container, or blob	Account (default), container, or blob	N/A

<sup>1</sup> For information about creating an account that supports using customer-managed keys with Queue storage, see [Create an account that supports customer-managed keys for queues](#).

<sup>2</sup> For information about creating an account that supports using customer-managed keys with Table storage, see [Create an account that supports customer-managed keys for tables](#).

#### NOTE

Microsoft-managed keys are rotated appropriately per compliance requirements. If you have specific key rotation requirements, Microsoft recommends that you move to customer-managed keys so that you can manage and audit the rotation yourself.

## Doubly encrypt data with infrastructure encryption

Customers who require high levels of assurance that their data is secure can also enable 256-bit AES encryption at the Azure Storage infrastructure level. When infrastructure encryption is enabled, data in a storage account is encrypted twice — once at the service level and once at the infrastructure level — with two different encryption algorithms and two different keys. Double encryption of Azure Storage data protects against a scenario where one of the encryption algorithms or keys may be compromised. In this scenario, the additional layer of encryption continues to protect your data.

Service-level encryption supports the use of either Microsoft-managed keys or customer-managed keys with Azure Key Vault. Infrastructure-level encryption relies on Microsoft-managed keys and always uses a separate key.

For more information about how to create a storage account that enables infrastructure encryption, see [Create a storage account with infrastructure encryption enabled for double encryption of data](#).

## Client-side encryption for blobs and queues

The Azure Blob Storage client libraries for .NET, Java, and Python support encrypting data within client applications before uploading to Azure Storage, and decrypting data while downloading to the client. The Queue Storage client libraries for .NET and Python also support client-side encryption.

### NOTE

Consider using the service-side encryption features provided by Azure Storage to protect your data, instead of client-side encryption.

The Blob Storage and Queue Storage client libraries uses [AES](#) in order to encrypt user data. There are two versions of client-side encryption available in the client libraries:

- Version 2 uses [Galois/Counter Mode \(GCM\)](#) mode with AES. The Blob Storage and Queue Storage SDKs support client-side encryption with v2.
- Version 1 uses [Cipher Block Chaining \(CBC\)](#) mode with AES. The Blob Storage, Queue Storage, and Table Storage SDKs support client-side encryption with v1.

### WARNING

Using client-side encryption v1 is no longer recommended due to a security vulnerability in the client library's implementation of CBC mode. For more information about this security vulnerability, see [Azure Storage updating client-side encryption in SDK to address security vulnerability](#). If you are currently using v1, we recommend that you update your application to use client-side encryption v2 and migrate your data.

The Azure Table Storage SDK supports only client-side encryption v1. Using client-side encryption with Table Storage is not recommended.

The following table shows which client libraries support which versions of client-side encryption and provides guidelines for migrating to client-side encryption v2.

CLIENT LIBRARY	VERSION OF CLIENT-SIDE ENCRYPTION SUPPORTED	RECOMMENDED MIGRATION	ADDITIONAL GUIDANCE
Blob Storage client libraries for .NET (version 12.13.0 and above), Java (version 12.18.0 and above), and Python (version 12.13.0 and above)	2.0  1.0 (for backward compatibility only)	Update your code to use client-side encryption v2.  Download any encrypted data to decrypt it, then reencrypt it with client-side encryption v2.	<a href="#">Client-side encryption for blobs</a>

CLIENT LIBRARY	VERSION OF CLIENT-SIDE ENCRYPTION SUPPORTED	RECOMMENDED MIGRATION	ADDITIONAL GUIDANCE
Blob Storage client library for .NET (version 12.12.0 and below), Java (version 12.17.0 and below), and Python (version 12.12.0 and below)	1.0 (not recommended)	<p>Update your application to use a version of the Blob Storage SDK that supports client-side encryption v2. See <a href="#">SDK support matrix for client-side encryption</a> for details.</p> <p>Update your code to use client-side encryption v2.</p> <p>Download any encrypted data to decrypt it, then reencrypt it with client-side encryption v2.</p>	<a href="#">Client-side encryption for blobs</a>
Queue Storage client library for .NET (version 12.11.0 and above) and Python (version 12.4 and above)	2.0 1.0 (for backward compatibility only)	Update your code to use client-side encryption v2.	<a href="#">Client-side encryption for queues</a>
Queue Storage client library for .NET (version 12.10.0 and below) and Python (version 12.3.0 and below)	1.0 (not recommended)	<p>Update your application to use a version of the Queue Storage SDK version that supports client-side encryption v2. See <a href="#">SDK support matrix for client-side encryption</a></p> <p>Update your code to use client-side encryption v2.</p>	<a href="#">Client-side encryption for queues</a>
Table Storage client library for .NET, Java, and Python	1.0 (not recommended)	Not available.	N/A

## Next steps

- [What is Azure Key Vault?](#)
- [Customer-managed keys for Azure Storage encryption](#)
- [Encryption scopes for Blob Storage](#)
- [Provide an encryption key on a request to Blob Storage](#)

# Customer-managed keys for Azure Storage encryption

8/22/2022 • 7 minutes to read • [Edit Online](#)

You can use your own encryption key to protect the data in your storage account. When you specify a customer-managed key, that key is used to protect and control access to the key that encrypts your data. Customer-managed keys offer greater flexibility to manage access controls.

You must use one of the following Azure key stores to store your customer-managed keys:

- [Azure Key Vault](#)
- [Azure Key Vault Managed Hardware Security Module \(HSM\)](#)

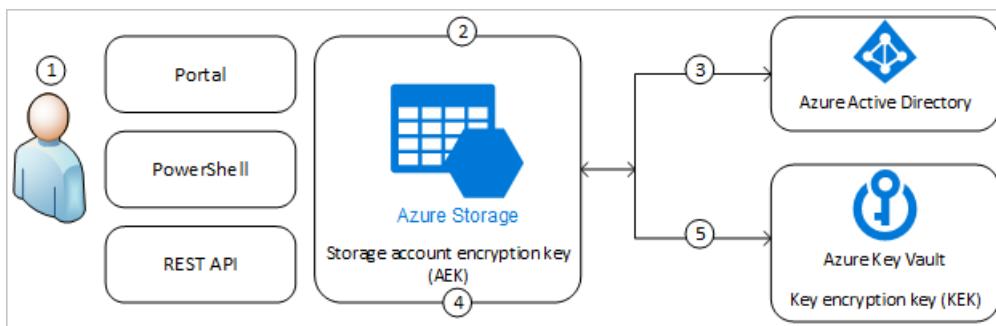
You can either create your own keys and store them in the key vault or managed HSM, or you can use the Azure Key Vault APIs to generate keys. The storage account and the key vault or managed HSM must be in the same Azure Active Directory (Azure AD) tenant, but they can be in different regions and subscriptions.

## NOTE

Azure Key Vault and Azure Key Vault Managed HSM support the same APIs and management interfaces for configuration.

## About customer-managed keys

The following diagram shows how Azure Storage uses Azure AD and a key vault or managed HSM to make requests using the customer-managed key:



The following list explains the numbered steps in the diagram:

1. An Azure Key Vault admin grants permissions to encryption keys to a managed identity. The managed identity may be either a user-assigned managed identity that you create and manage, or a system-assigned managed identity that is associated with the storage account.
2. An Azure Storage admin configures encryption with a customer-managed key for the storage account.
3. Azure Storage uses the managed identity to which the Azure Key Vault admin granted permissions in step 1 to authenticate access to Azure Key Vault via Azure AD.
4. Azure Storage wraps the account encryption key with the customer-managed key in Azure Key Vault.
5. For read/write operations, Azure Storage sends requests to Azure Key Vault to unwrap the account encryption key to perform encryption and decryption operations.

The managed identity that is associated with the storage account must have these permissions at a minimum to

access a customer-managed key in Azure Key Vault:

- *wrapkey*
- *unwrapkey*
- *get*

For more information about key permissions, see [Key types, algorithms, and operations](#).

Azure Policy provides a built-in policy to require that storage accounts use customer-managed keys for Blob Storage and Azure Files workloads. For more information, see the **Storage** section in [Azure Policy built-in policy definitions](#).

## Customer-managed keys for queues and tables

Data stored in Queue and Table storage is not automatically protected by a customer-managed key when customer-managed keys are enabled for the storage account. You can optionally configure these services to be included in this protection at the time that you create the storage account.

For more information about how to create a storage account that supports customer-managed keys for queues and tables, see [Create an account that supports customer-managed keys for tables and queues](#).

Data in Blob storage and Azure Files is always protected by customer-managed keys when customer-managed keys are configured for the storage account.

## Enable customer-managed keys for a storage account

When you configure a customer-managed key, Azure Storage wraps the root data encryption key for the account with the customer-managed key in the associated key vault or managed HSM. Enabling customer-managed keys does not impact performance, and takes effect immediately.

When you enable or disable customer-managed keys, or when you modify the key or the key version, the protection of the root encryption key changes, but the data in your Azure Storage account does not need to be re-encrypted.

You can enable customer-managed keys on both new and existing storage accounts. When you enable customer-managed keys, you must specify a managed identity to be used to authorize access to the key vault that contains the key. The managed identity may be either a user-assigned or system-assigned managed identity:

- When you configure customer-managed keys at the time that you create a storage account, you must use a user-assigned managed identity.
- When you configure customer-managed keys on an existing storage account, you can use either a user-assigned managed identity or a system-assigned managed identity.

To learn more about system-assigned versus user-assigned managed identities, see [Managed identities for Azure resources](#).

You can switch between customer-managed keys and Microsoft-managed keys at any time. For more information about Microsoft-managed keys, see [About encryption key management](#).

To learn how to configure Azure Storage encryption with customer-managed keys in a key vault, see [Configure encryption with customer-managed keys stored in Azure Key Vault](#). To configure customer-managed keys in a managed HSM, see [Configure encryption with customer-managed keys stored in Azure Key Vault Managed HSM](#).

#### **IMPORTANT**

Customer-managed keys rely on managed identities for Azure resources, a feature of Azure AD. Managed identities do not currently support cross-tenant scenarios. When you configure customer-managed keys in the Azure portal, a managed identity is automatically assigned to your storage account under the covers. If you subsequently move the subscription, resource group, or storage account from one Azure AD tenant to another, the managed identity associated with the storage account is not transferred to the new tenant, so customer-managed keys may no longer work. For more information, see [Transferring a subscription between Azure AD directories](#) in [FAQs and known issues with managed identities for Azure resources](#).

Azure storage encryption supports RSA and RSA-HSM keys of sizes 2048, 3072 and 4096. For more information about keys, see [About keys](#).

Using a key vault or managed HSM has associated costs. For more information, see [Key Vault pricing](#).

## Update the key version

When you configure encryption with customer-managed keys, you have two options for updating the key version:

- **Automatically update the key version:** To automatically update a customer-managed key when a new version is available, omit the key version when you enable encryption with customer-managed keys for the storage account. If the key version is omitted, then Azure Storage checks the key vault or managed HSM daily for a new version of a customer-managed key. If a new key version is available, then Azure Storage automatically uses the latest version of the key.  
  
Azure Storage checks the key vault for a new key version only once daily. When you rotate a key, be sure to wait 24 hours before disabling the older version.
- **Manually update the key version:** To use a specific version of a key for Azure Storage encryption, specify that key version when you enable encryption with customer-managed keys for the storage account. If you specify the key version, then Azure Storage uses that version for encryption until you manually update the key version.

When the key version is explicitly specified, then you must manually update the storage account to use the new key version URI when a new version is created. To learn how to update the storage account to use a new version of the key, see [Configure encryption with customer-managed keys stored in Azure Key Vault](#) or [Configure encryption with customer-managed keys stored in Azure Key Vault Managed HSM](#).

When you update the key version, the protection of the root encryption key changes, but the data in your Azure Storage account is not re-encrypted. There is no further action required from the user.

#### **NOTE**

To rotate a key, create a new version of the key in the key vault or managed HSM, according to your compliance policies. You can rotate your key manually or create a function to rotate it on a schedule.

## Revoke access to customer-managed keys

You can revoke the storage account's access to the customer-managed key at any time. After access to customer-managed keys is revoked, or after the key has been disabled or deleted, clients cannot call operations that read from or write to a blob or its metadata. Attempts to call any of the following operations will fail with error code 403 (Forbidden) for all users:

- [List Blobs](#), when called with the `include=metadata` parameter on the request URI

- [Get Blob](#)
- [Get Blob Properties](#)
- [Get Blob Metadata](#)
- [Set Blob Metadata](#)
- [Snapshot Blob](#), when called with the `x-ms-meta-name` request header
- [Copy Blob](#)
- [Copy Blob From URL](#)
- [Set Blob Tier](#)
- [Put Block](#)
- [Put Block From URL](#)
- [Append Block](#)
- [Append Block From URL](#)
- [Put Blob](#)
- [Put Page](#)
- [Put Page From URL](#)
- [Incremental Copy Blob](#)

To call these operations again, restore access to the customer-managed key.

All data operations that are not listed in this section may proceed after customer-managed keys are revoked or a key is disabled or deleted.

To revoke access to customer-managed keys, use [PowerShell](#) or [Azure CLI](#).

## Customer-managed keys for Azure managed disks

Customer-managed keys are also available for managing encryption of Azure managed disks. Customer-managed keys behave differently for managed disks than for Azure Storage resources. For more information, see [Server-side encryption of Azure managed disks](#) for Windows or [Server side encryption of Azure managed disks](#) for Linux.

## Next steps

- [Azure Storage encryption for data at rest](#)
- [Configure encryption with customer-managed keys stored in Azure Key Vault](#)
- [Configure encryption with customer-managed keys stored in Azure Key Vault Managed HSM](#)

# Provide an encryption key on a request to Blob storage

8/22/2022 • 3 minutes to read • [Edit Online](#)

Clients making requests against Azure Blob storage can provide an AES-256 encryption key to encrypt that blob on a write operation. Subsequent requests to read or write to the blob must include the same key. Including the encryption key on the request provides granular control over encryption settings for Blob storage operations. Customer-provided keys can be stored in Azure Key Vault or in another key store.

## Encrypting read and write operations

When a client application provides an encryption key on the request, Azure Storage performs encryption and decryption transparently while reading and writing blob data. Azure Storage writes an SHA-256 hash of the encryption key alongside the blob's contents. The hash is used to verify that all subsequent operations against the blob use the same encryption key.

Azure Storage doesn't store or manage the encryption key that the client sends with the request. The key is securely discarded as soon as the encryption or decryption process is complete.

When a client creates or updates a blob using a customer-provided key on the request, then subsequent read and write requests for that blob must also provide the key. If the key isn't provided on a request for a blob that has already been encrypted with a customer-provided key, then the request fails with error code 409 (Conflict).

If the client application sends an encryption key on the request, and the storage account is also encrypted using a Microsoft-managed key or a customer-managed key, then Azure Storage uses the key provided on the request for encryption and decryption.

To send the encryption key as part of the request, a client must establish a secure connection to Azure Storage using HTTPS.

Each blob snapshot or blob version can have its own encryption key.

Object replication isn't supported for blobs in the source account that are encrypted with a customer-provided key.

## Request headers for specifying customer-provided keys

For REST calls, clients can use the following headers to securely pass encryption key information on a request to Blob storage:

REQUEST HEADER	DESCRIPTION
<code>x-ms-encryption-key</code>	Required for both write and read requests. A Base64-encoded AES-256 encryption key value.
<code>x-ms-encryption-key-sha256</code>	Required for both write and read requests. The Base64-encoded SHA256 of the encryption key.
<code>x-ms-encryption-algorithm</code>	Required for write requests, optional for read requests. Specifies the algorithm to use when encrypting data using the given key. The value of this header must be <code>AES256</code> .

Specifying encryption keys on the request is optional. However, if you specify one of the headers listed above for a write operation, then you must specify all of them.

## Blob storage operations supporting customer-provided keys

The following Blob storage operations support sending customer-provided encryption keys on a request:

- [Put Blob](#)
- [Put Block List](#)
- [Put Block](#)
- [Put Block from URL](#)
- [Put Page](#)
- [Put Page from URL](#)
- [Append Block](#)
- [Set Blob Properties](#)
- [Set Blob Metadata](#)
- [Get Blob](#)
- [Get Blob Properties](#)
- [Get Blob Metadata](#)
- [Snapshot Blob](#)

## Rotate customer-provided keys

To rotate an encryption key that was used to encrypt a blob, download the blob and then reupload it with the new encryption key.

### IMPORTANT

The Azure portal cannot be used to read from or write to a container or blob that is encrypted with a key provided on the request.

Be sure to protect the encryption key that you provide on a request to Blob storage in a secure key store like Azure Key Vault. If you attempt a write operation on a container or blob without the encryption key, the operation will fail, and you will lose access to the object.

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## Next steps

- [Specify a customer-provided key on a request to Blob storage with .NET](#)
- [Azure Storage encryption for data at rest](#)

# Encryption scopes for Blob storage

8/22/2022 • 5 minutes to read • [Edit Online](#)

Encryption scopes enable you to manage encryption with a key that is scoped to a container or an individual blob. You can use encryption scopes to create secure boundaries between data that resides in the same storage account but belongs to different customers.

For more information about working with encryption scopes, see [Create and manage encryption scopes](#).

## How encryption scopes work

By default, a storage account is encrypted with a key that is scoped to the entire storage account. When you define an encryption scope, you specify a key that may be scoped to a container or an individual blob. When the encryption scope is applied to a blob, the blob is encrypted with that key. When the encryption scope is applied to a container, it serves as the default scope for blobs in that container, so that all blobs that are uploaded to that container may be encrypted with the same key. The container can be configured to enforce the default encryption scope for all blobs in the container, or to permit an individual blob to be uploaded to the container with an encryption scope other than the default.

Read operations on a blob that was created with an encryption scope happen transparently, so long as the encryption scope is not disabled.

### Key management

When you define an encryption scope, you can specify whether the scope is protected with a Microsoft-managed key or with a customer-managed key that is stored in Azure Key Vault. Different encryption scopes on the same storage account can use either Microsoft-managed or customer-managed keys. You can also switch the type of key used to protect an encryption scope from a customer-managed key to a Microsoft-managed key, or vice versa, at any time. For more information about customer-managed keys, see [Customer-managed keys for Azure Storage encryption](#). For more information about Microsoft-managed keys, see [About encryption key management](#).

If you define an encryption scope with a customer-managed key, then you can choose to update the key version either automatically or manually. If you choose to automatically update the key version, then Azure Storage checks the key vault or managed HSM daily for a new version of the customer-managed key and automatically updates the key to the latest version. For more information about updating the key version for a customer-managed key, see [Update the key version](#).

Azure Policy provides a built-in policy to require that encryption scopes use customer-managed keys. For more information, see the **Storage** section in [Azure Policy built-in policy definitions](#).

A storage account may have up to 10,000 encryption scopes that are protected with customer-managed keys for which the key version is automatically updated. If your storage account already has 10,000 encryption scopes that are protected with customer-managed keys that are being automatically updated, then the key version must be updated manually for any additional encryption scopes that are protected with customer-managed keys.

### Infrastructure encryption

Infrastructure encryption in Azure Storage enables double encryption of data. With infrastructure encryption, data is encrypted twice — once at the service level and once at the infrastructure level — with two different encryption algorithms and two different keys.

Infrastructure encryption is supported for an encryption scope, as well as at the level of the storage account. If

infrastructure encryption is enabled for an account, then any encryption scope created on that account automatically uses infrastructure encryption. If infrastructure encryption is not enabled at the account level, then you have the option to enable it for an encryption scope at the time that you create the scope. The infrastructure encryption setting for an encryption scope cannot be changed after the scope is created.

For more information about infrastructure encryption, see [Enable infrastructure encryption for double encryption of data](#).

### Encryption scopes for containers and blobs

When you create a container, you can specify a default encryption scope for the blobs that are subsequently uploaded to that container. When you specify a default encryption scope for a container, you can decide how the default encryption scope is enforced:

- You can require that all blobs uploaded to the container use the default encryption scope. In this case, every blob in the container is encrypted with the same key.
- You can permit a client to override the default encryption scope for the container, so that a blob may be uploaded with an encryption scope other than the default scope. In this case, the blobs in the container may be encrypted with different keys.

The following table summarizes the behavior of a blob upload operation, depending on how the default encryption scope is configured for the container:

THE ENCRYPTION SCOPE DEFINED ON THE CONTAINER IS...	UPLOADING A BLOB WITH THE DEFAULT ENCRYPTION SCOPE...	UPLOADING A BLOB WITH AN ENCRYPTION SCOPE OTHER THAN THE DEFAULT SCOPE...
A default encryption scope with overrides permitted	Succeeds	Succeeds
A default encryption scope with overrides prohibited	Succeeds	Fails

A default encryption scope must be specified for a container at the time that the container is created.

If no default encryption scope is specified for the container, then you can upload a blob using any encryption scope that you've defined for the storage account. The encryption scope must be specified at the time that the blob is uploaded.

When you upload a new blob with an encryption scope, you cannot change the default access tier for that blob. You also cannot change the access tier for an existing blob that uses an encryption scope. For more information about access tiers, see [Hot, Cool, and Archive access tiers for blob data](#).

## Disabling an encryption scope

When you disable an encryption scope, any subsequent read or write operations made with the encryption scope will fail with HTTP error code 403 (Forbidden). If you re-enable the encryption scope, read and write operations will proceed normally again.

When an encryption scope is disabled, you are no longer billed for it. Disable any encryption scopes that are not needed to avoid unnecessary charges.

If your encryption scope is protected with a customer-managed key, and you revoke the key in the key vault, the data will become inaccessible. Be sure to disable the encryption scope prior to revoking the key in key vault to avoid being charged for the encryption scope.

Keep in mind that customer-managed keys are protected by soft delete and purge protection in the key vault, and a deleted key is subject to the behavior defined for by those properties. For more information, see one of

the following topics in the Azure Key Vault documentation:

- [How to use soft-delete with PowerShell](#)
- [How to use soft-delete with CLI](#)

**IMPORTANT**

It is not possible to delete an encryption scope.

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## Next steps

- [Azure Storage encryption for data at rest](#)
- [Create and manage encryption scopes](#)
- [Customer-managed keys for Azure Storage encryption](#)
- [What is Azure Key Vault?](#)

# Use private endpoints for Azure Storage

8/22/2022 • 8 minutes to read • [Edit Online](#)

You can use [private endpoints](#) for your Azure Storage accounts to allow clients on a virtual network (VNet) to securely access data over a [Private Link](#). The private endpoint uses a separate IP address from the VNet address space for each storage account service. Network traffic between the clients on the VNet and the storage account traverses over the VNet and a private link on the Microsoft backbone network, eliminating exposure from the public internet.

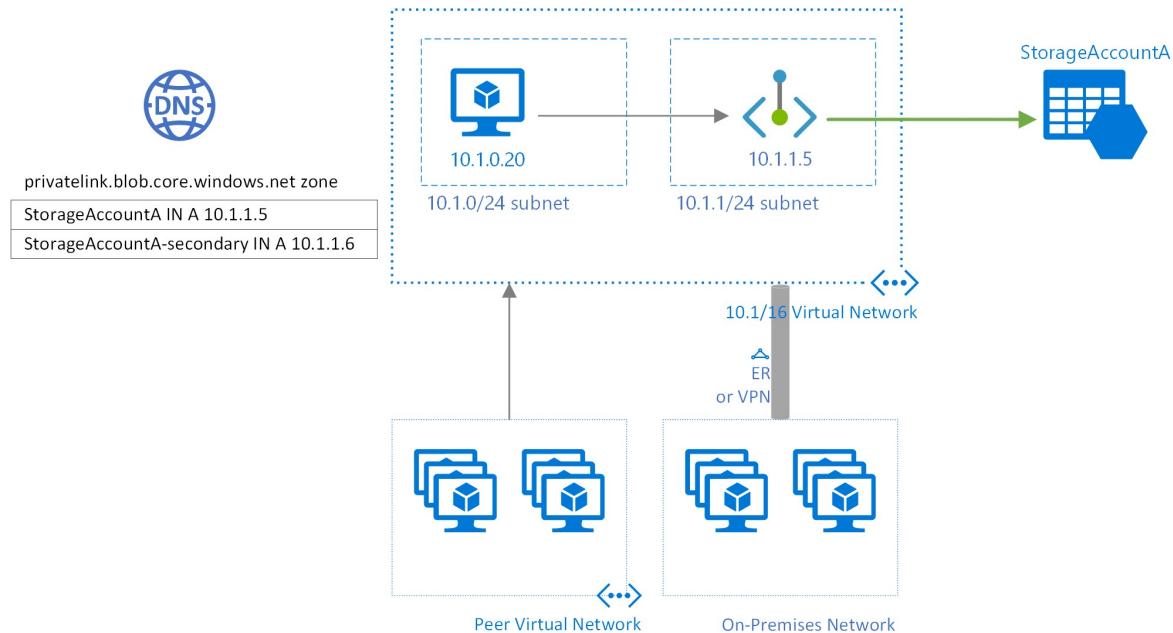
## NOTE

Private endpoints are not available for general-purpose v1 storage accounts.

Using private endpoints for your storage account enables you to:

- Secure your storage account by configuring the storage firewall to block all connections on the public endpoint for the storage service.
- Increase security for the virtual network (VNet), by enabling you to block exfiltration of data from the VNet.
- Securely connect to storage accounts from on-premises networks that connect to the VNet using [VPN](#) or [ExpressRoutes](#) with private-peering.

## Conceptual overview



A private endpoint is a special network interface for an Azure service in your [Virtual Network](#) (VNet). When you create a private endpoint for your storage account, it provides secure connectivity between clients on your VNet and your storage. The private endpoint is assigned an IP address from the IP address range of your VNet. The connection between the private endpoint and the storage service uses a secure private link.

Applications in the VNet can connect to the storage service over the private endpoint seamlessly, **using the same connection strings and authorization mechanisms that they would use otherwise**. Private endpoints can be used with all protocols supported by the storage account, including REST and SMB.

Private endpoints can be created in subnets that use [Service Endpoints](#). Clients in a subnet can thus connect to one storage account using private endpoint, while using service endpoints to access others.

When you create a private endpoint for a storage service in your VNet, a consent request is sent for approval to the storage account owner. If the user requesting the creation of the private endpoint is also an owner of the storage account, this consent request is automatically approved.

Storage account owners can manage consent requests and the private endpoints through the '*Private endpoints*' tab for the storage account in the [Azure portal](#).

**TIP**

If you want to restrict access to your storage account through the private endpoint only, configure the storage firewall to deny or control access through the public endpoint.

You can secure your storage account to only accept connections from your VNet by [configuring the storage firewall](#) to deny access through its public endpoint by default. You don't need a firewall rule to allow traffic from a VNet that has a private endpoint, since the storage firewall only controls access through the public endpoint. Private endpoints instead rely on the consent flow for granting subnets access to the storage service.

**NOTE**

When copying blobs between storage accounts, your client must have network access to both accounts. So if you choose to use a private link for only one account (either the source or the destination), make sure that your client has network access to the other account. To learn about other ways to configure network access, see [Configure Azure Storage firewalls and virtual networks](#).

## Creating a private endpoint

To create a private endpoint by using the Azure Portal, see [Connect privately to a storage account from the Storage Account experience in the Azure portal](#).

To create a private endpoint by using PowerShell or the Azure CLI, see either of these articles. Both of them feature an Azure web app as the target service, but the steps to create a private link are the same for an Azure Storage account.

- [Create a private endpoint using Azure CLI](#)
- [Create a private endpoint using Azure PowerShell](#)

When you create a private endpoint, you must specify the storage account and the storage service to which it connects.

You need a separate private endpoint for each storage resource that you need to access, namely [Blobs](#), [Data Lake Storage Gen2](#), [Files](#), [Queues](#), [Tables](#), or [Static Websites](#). On the private endpoint, these storage services are defined as the **target sub-resource** of the associated storage account.

If you create a private endpoint for the Data Lake Storage Gen2 storage resource, then you should also create one for the Blob storage resource. That's because operations that target the Data Lake Storage Gen2 endpoint might be redirected to the Blob endpoint. By creating a private endpoint for both resources, you ensure that operations can complete successfully.

**TIP**

Create a separate private endpoint for the secondary instance of the storage service for better read performance on RA-GRS accounts. Make sure to create a general-purpose v2(Standard or Premium) storage account.

For read access to the secondary region with a storage account configured for geo-redundant storage, you need separate private endpoints for both the primary and secondary instances of the service. You don't need to create a private endpoint for the secondary instance for **failover**. The private endpoint will automatically connect to

the new primary instance after failover. For more information about storage redundancy options, see [Azure Storage redundancy](#).

## Connecting to a private endpoint

Clients on a VNet using the private endpoint should use the same connection string for the storage account as clients connecting to the public endpoint. We rely upon DNS resolution to automatically route the connections from the VNet to the storage account over a private link.

### IMPORTANT

Use the same connection string to connect to the storage account using private endpoints as you'd use otherwise. Please don't connect to the storage account using its `privatelink` subdomain URL.

By default, we create a [private DNS zone](#) attached to the VNet with the necessary updates for the private endpoints. However, if you're using your own DNS server, you may need to make additional changes to your DNS configuration. The section on [DNS changes](#) below describes the updates required for private endpoints.

## DNS changes for private endpoints

When you create a private endpoint, the DNS CNAME resource record for the storage account is updated to an alias in a subdomain with the prefix `privatelink`. By default, we also create a [private DNS zone](#), corresponding to the `privatelink` subdomain, with the DNS A resource records for the private endpoints.

When you resolve the storage endpoint URL from outside the VNet with the private endpoint, it resolves to the public endpoint of the storage service. When resolved from the VNet hosting the private endpoint, the storage endpoint URL resolves to the private endpoint's IP address.

For the illustrated example above, the DNS resource records for the storage account 'StorageAccountA', when resolved from outside the VNet hosting the private endpoint, will be:

NAME	TYPE	VALUE
<code>StorageAccountA.blob.core.windows.net</code>	CNAME	<code>StorageAccountA.privatelink.blob.core.windows.net</code>
<code>StorageAccountA.privatelink.blob.core.windows.net</code>	CNAME	<storage service public endpoint>
<storage service public endpoint>	A	<storage service public IP address>

As previously mentioned, you can deny or control access for clients outside the VNet through the public endpoint using the storage firewall.

The DNS resource records for StorageAccountA, when resolved by a client in the VNet hosting the private endpoint, will be:

NAME	TYPE	VALUE
<code>StorageAccountA.blob.core.windows.net</code>	CNAME	<code>StorageAccountA.privatelink.blob.core.windows.net</code>
<code>StorageAccountA.privatelink.blob.core.windows.net</code>	A	10.1.1.5

This approach enables access to the storage account **using the same connection string** for clients on the VNet hosting the private endpoints, as well as clients outside the VNet.

If you are using a custom DNS server on your network, clients must be able to resolve the FQDN for the storage account endpoint to the private endpoint IP address. You should configure your DNS server to delegate your private link subdomain to the private DNS zone for the VNet, or configure the A records for

`StorageAccountA.privatelink.blob.core.windows.net` with the private endpoint IP address.

#### TIP

When using a custom or on-premises DNS server, you should configure your DNS server to resolve the storage account name in the `privatelink` subdomain to the private endpoint IP address. You can do this by delegating the `privatelink` subdomain to the private DNS zone of the VNet or by configuring the DNS zone on your DNS server and adding the DNS A records.

The recommended DNS zone names for private endpoints for storage services, and the associated endpoint target sub-resources, are:

STORAGE SERVICE	TARGET SUB-RESOURCE	ZONE NAME
Blob service	blob	<code>privatelink.blob.core.windows.net</code>
Data Lake Storage Gen2	dfs	<code>privatelink.dfs.core.windows.net</code>
File service	file	<code>privatelink.file.core.windows.net</code>
Queue service	queue	<code>privatelink.queue.core.windows.net</code>
Table service	table	<code>privatelink.table.core.windows.net</code>
Static Websites	web	<code>privatelink.web.core.windows.net</code>

For more information on configuring your own DNS server to support private endpoints, refer to the following articles:

- [Name resolution for resources in Azure virtual networks](#)
- [DNS configuration for private endpoints](#)

## Pricing

For pricing details, see [Azure Private Link pricing](#).

## Known Issues

Keep in mind the following known issues about private endpoints for Azure Storage.

### Storage access constraints for clients in VNets with private endpoints

Clients in VNets with existing private endpoints face constraints when accessing other storage accounts that have private endpoints. For example, suppose a VNet N1 has a private endpoint for a storage account A1 for Blob storage. If storage account A2 has a private endpoint in a VNet N2 for Blob storage, then clients in VNet N1 must also access Blob storage in account A2 using a private endpoint. If storage account A2 does not have any private endpoints for Blob storage, then clients in VNet N1 can access Blob storage in that account without a private endpoint.

This constraint is a result of the DNS changes made when account A2 creates a private endpoint.

### Network Security Group rules for subnets with private endpoints

Currently, you can't configure [Network Security Group](#) (NSG) rules and user-defined routes for private endpoints. NSG rules applied to the subnet hosting the private endpoint are not applied to the private endpoint. They are applied only to other endpoints (For example: network interface controllers). A limited workaround for this issue is to implement your access rules for private endpoints on the source subnets, though this approach may require a higher management overhead.

## **Copying blobs between storage accounts**

You can copy blobs between storage accounts by using private endpoints only if you use the Azure REST API, or tools that use the REST API. These tools include AzCopy, Storage Explorer, Azure PowerShell, Azure CLI, and the Azure Blob Storage SDKs.

Only private endpoints that target the Blob storage resource are supported. Private endpoints that target the Data Lake Storage Gen2 or the File resource are not yet supported. Also, copying between storage accounts by using the Network File System (NFS) protocol is not yet supported.

## Next steps

- [Configure Azure Storage firewalls and virtual networks](#)
- [Security recommendations for Blob storage](#)

# Network routing preference for Azure Storage

8/22/2022 • 3 minutes to read • [Edit Online](#)

You can configure network [routing preference](#) for your Azure storage account to specify how network traffic is routed to your account from clients over the internet. By default, traffic from the internet is routed to the public endpoint of your storage account over the [Microsoft global network](#). Azure Storage provides additional options for configuring how traffic is routed to your storage account.

Configuring routing preference gives you the flexibility to optimize your traffic either for premium network performance or for cost. When you configure a routing preference, you specify how traffic will be directed to the public endpoint for your storage account by default. You can also publish route-specific endpoints for your storage account.

## NOTE

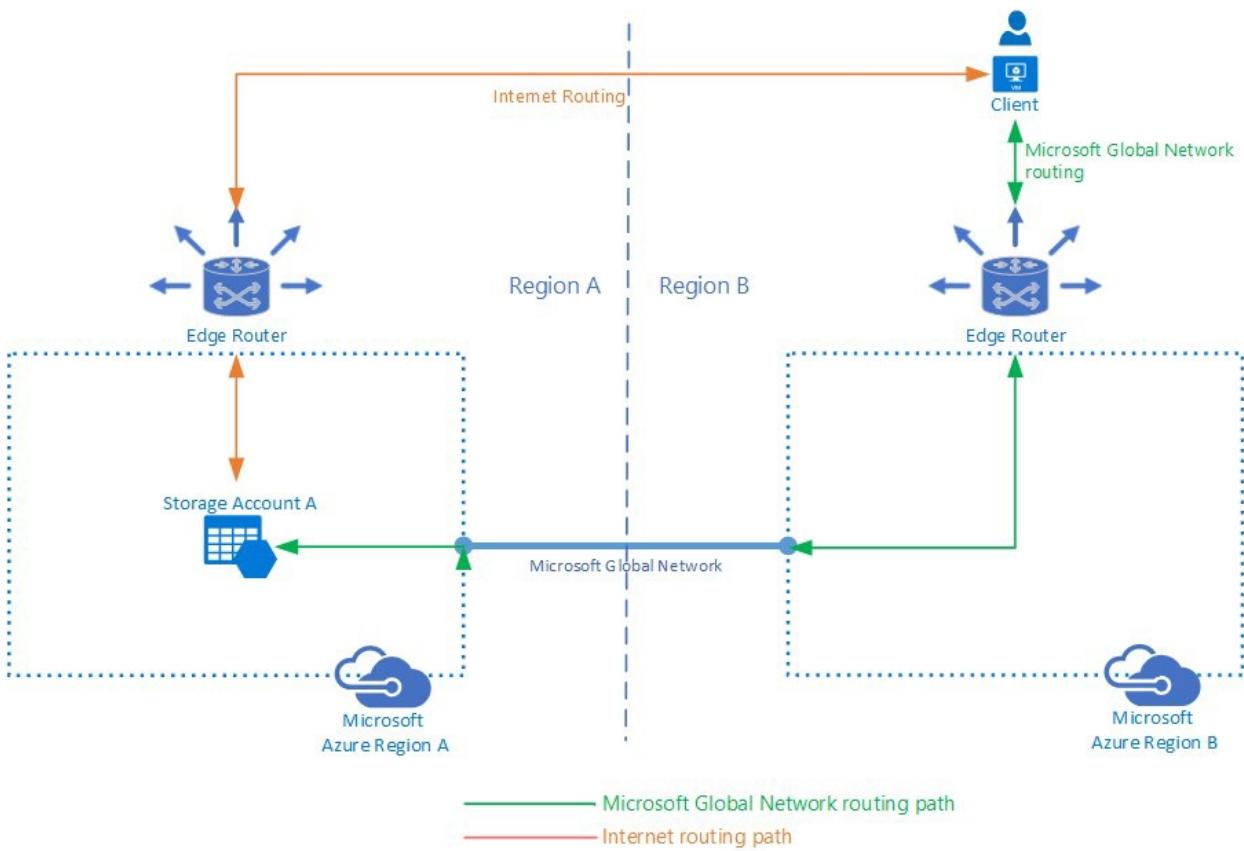
This feature is not supported in premium performance storage accounts or accounts configured to use Zone-redundant storage (ZRS).

## Microsoft global network versus Internet routing

By default, clients outside of the Azure environment access your storage account over the Microsoft global network. The Microsoft global network is optimized for low-latency path selection to deliver premium network performance with high reliability. Both inbound and outbound traffic are routed through the point of presence (POP) that is closest to the client. This default routing configuration ensures that traffic to and from your storage account traverses over the Microsoft global network for the bulk of its path, maximizing network performance.

You can change the routing configuration for your storage account so that both inbound and outbound traffic are routed to and from clients outside of the Azure environment through the POP closest to the storage account. This route minimizes the traversal of your traffic over the Microsoft global network, handing it off to the transit ISP at the earliest opportunity. Utilizing this routing configuration lowers networking costs.

The following diagram shows how traffic flows between the client and the storage account for each routing preference:



For more information on routing preference in Azure, see [What is routing preference?](#).

## Routing configuration

For step-by-step guidance that shows you how to configure the routing preference and route-specific endpoints, see [Configure network routing preference for Azure Storage](#).

You can choose between the Microsoft global network and internet routing as the default routing preference for the public endpoint of your storage account. The default routing preference applies to all traffic from clients outside Azure and affects the endpoints for Azure Data Lake Storage Gen2, Blob storage, Azure Files, and static websites. Configuring routing preference is not supported for Azure Queues or Azure Tables.

You can also publish route-specific endpoints for your storage account. When you publish route-specific endpoints, Azure Storage creates new public endpoints for your storage account that route traffic over the desired path. This flexibility enables you to direct traffic to your storage account over a specific route without changing your default routing preference.

For example, publishing an internet route-specific endpoint for the 'StorageAccountA' will publish the following endpoints for your storage account:

STORAGE SERVICE	ROUTE-SPECIFIC ENDPOINT
Blob service	StorageAccountA-internetworking.blob.core.windows.net
Data Lake Storage Gen2	StorageAccountA-internetworking.dfs.core.windows.net
File service	StorageAccountA-internetworking.file.core.windows.net

STORAGE SERVICE	ROUTE-SPECIFIC ENDPOINT
Static Websites	StorageAccountA-internetrouting.web.core.windows.net

If you have a read-access geo-redundant storage (RA-GRS) or a read-access geo-zone-redundant storage (RA-GZRS) storage account, publishing route-specific endpoints also automatically creates the corresponding endpoints in the secondary region for read access.

STORAGE SERVICE	ROUTE-SPECIFIC READ-ONLY SECONDARY ENDPOINT
Blob service	StorageAccountA-internetrouting-secondary.blob.core.windows.net
Data Lake Storage Gen2	StorageAccountA-internetrouting-secondary.dfs.core.windows.net
File service	StorageAccountA-internetrouting-secondary.file.core.windows.net
Static Websites	StorageAccountA-internetrouting-secondary.web.core.windows.net

The connection strings for the published route-specific endpoints can be copied via the [Azure portal](#). These connection strings can be used for Shared Key authorization with all existing Azure Storage SDKs and APIs.

## Regional availability

Routing preference for Azure Storage is available in the following regions:

- Central US
- Central US EUAP
- East US
- East US 2
- East US 2
- East US 2 EUAP
- South Central US
- West Central US
- West US
- West US 2
- France Central
- France South
- Germany North
- Germany West Central
- North Central US
- North Europe
- Norway East
- Switzerland North
- Switzerland West
- UK South
- UK West
- West Europe

- UAE Central
- East Asia
- Southeast Asia
- Japan East
- Japan West
- West India
- Australia East
- Australia Southeast

The following known issues affect the routing preference for Azure Storage:

- Access requests for the route-specific endpoint for the Microsoft global network fail with HTTP error 404 or equivalent. Routing over the Microsoft global network works as expected when it is set as the default routing preference for the public endpoint.

## Pricing and billing

For pricing and billing details, see the **Pricing** section in [What is routing preference?](#).

## Next steps

- [What is routing preference?](#)
- [Configure network routing preference](#)
- [Configure Azure Storage firewalls and virtual networks](#)
- [Security recommendations for Blob storage](#)

# Data protection overview

8/22/2022 • 11 minutes to read • [Edit Online](#)

Azure Storage provides data protection for Blob Storage and Azure Data Lake Storage Gen2 to help you to prepare for scenarios where you need to recover data that has been deleted or overwritten. It's important to think about how to best protect your data before an incident occurs that could compromise it. This guide can help you decide in advance which data protection features your scenario requires, and how to implement them. If you should need to recover data that has been deleted or overwritten, this overview also provides guidance on how to proceed, based on your scenario.

In the Azure Storage documentation, *data protection* refers to strategies for protecting the storage account and data within it from being deleted or modified, or for restoring data after it has been deleted or modified. Azure Storage also offers options for *disaster recovery*, including multiple levels of redundancy to protect your data from service outages due to hardware problems or natural disasters, and customer-managed failover in the event that the data center in the primary region becomes unavailable. For more information about how your data is protected from service outages, see [Disaster recovery](#).

## Recommendations for basic data protection

If you are looking for basic data protection coverage for your storage account and the data that it contains, then Microsoft recommends taking the following steps to begin with:

- Configure an Azure Resource Manager lock on the storage account to protect the account from deletion or configuration changes. [Learn more...](#)
- Enable container soft delete for the storage account to recover a deleted container and its contents. [Learn more...](#)
- Save the state of a blob at regular intervals:
  - For Blob Storage workloads, enable blob versioning to automatically save the state of your data each time a blob is overwritten. [Learn more...](#)
  - For Azure Data Lake Storage workloads, take manual snapshots to save the state of your data at a particular point in time. [Learn more...](#)

These options, as well as additional data protection options for other scenarios, are described in more detail in the following section.

For an overview of the costs involved with these features, see [Summary of cost considerations](#).

## Overview of data protection options

The following table summarizes the options available in Azure Storage for common data protection scenarios. Choose the scenarios that are applicable to your situation to learn more about the options available to you. Note that not all features are available at this time for storage accounts with a hierarchical namespace enabled.

SCENARIO	DATA PROTECTION OPTION	RECOMMENDATIONS	PROTECTION BENEFIT	AVAILABLE FOR DATA LAKE STORAGE
----------	------------------------	-----------------	--------------------	---------------------------------

SCENARIO	DATA PROTECTION OPTION	RECOMMENDATIONS	PROTECTION BENEFIT	AVAILABLE FOR DATA LAKE STORAGE
Prevent a storage account from being deleted or modified.	Azure Resource Manager lock <a href="#">Learn more...</a>	Lock all of your storage accounts with an Azure Resource Manager lock to prevent deletion of the storage account.	Protects the storage account against deletion or configuration changes.  Does not protect containers or blobs in the account from being deleted or overwritten.	Yes
Prevent a blob version from being deleted for an interval that you control.	Immutability policy on a blob version <a href="#">Learn more...</a>	Set an immutability policy on an individual blob version to protect business-critical documents, for example, in order to meet legal or regulatory compliance requirements.	Protects a blob version from being deleted and its metadata from being overwritten. An overwrite operation creates a new version.  If at least one container has version-level immutability enabled, the storage account is also protected from deletion. Container deletion fails if at least one blob exists in the container.	No
Prevent a container and its blobs from being deleted or modified for an interval that you control.	Immutability policy on a container <a href="#">Learn more...</a>	Set an immutability policy on a container to protect business-critical documents, for example, in order to meet legal or regulatory compliance requirements.	Protects a container and its blobs from all deletes and overwrites.  When a legal hold or a locked time-based retention policy is in effect, the storage account is also protected from deletion. Containers for which no immutability policy has been set are not protected from deletion.	Yes, in preview

SCENARIO	DATA PROTECTION OPTION <a href="#">Learn more...</a>	RECOMMENDATIONS	PROTECTION BENEFIT	AVAILABLE FOR DATA LAKE STORAGE
Restore a deleted container within a specified interval.	Container soft delete <a href="#">Learn more...</a>	<p>Enable container soft delete for all storage accounts, with a minimum retention interval of 7 days.</p> <p>Enable blob versioning and blob soft delete together with container soft delete to protect individual blobs in a container.</p> <p>Store containers that require different retention periods in separate storage accounts.</p>	<p>A deleted container and its contents may be restored within the retention period.</p> <p>Only container-level operations (e.g., <a href="#">Delete Container</a>) can be restored.</p> <p>Container soft delete does not enable you to restore an individual blob in the container if that blob is deleted.</p>	Yes
Automatically save the state of a blob in a previous version when it is overwritten.	Blob versioning <a href="#">Learn more...</a>	<p>Enable blob versioning, together with container soft delete and blob soft delete, for storage accounts where you need optimal protection for blob data.</p> <p>Store blob data that does not require versioning in a separate account to limit costs.</p>	Every blob write operation creates a new version. The current version of a blob may be restored from a previous version if the current version is deleted or overwritten.	No
Restore a deleted blob or blob version within a specified interval.	Blob soft delete <a href="#">Learn more...</a>	<p>Enable blob soft delete for all storage accounts, with a minimum retention interval of 7 days.</p> <p>Enable blob versioning and container soft delete together with blob soft delete for optimal protection of blob data.</p> <p>Store blobs that require different retention periods in separate storage accounts.</p>	A deleted blob or blob version may be restored within the retention period.	Yes

SCENARIO	DATA PROTECTION OPTION	RECOMMENDATIONS	PROTECTION BENEFIT	AVAILABLE FOR DATA LAKE STORAGE
Restore a set of block blobs to a previous point in time.	Point-in-time restore <a href="#">Learn more...</a>	To use point-in-time restore to revert to an earlier state, design your application to delete individual block blobs rather than deleting containers.	A set of block blobs may be reverted to their state at a specific point in the past.  Only operations performed on block blobs are reverted. Any operations performed on containers, page blobs, or append blobs are not reverted.	No
Manually save the state of a blob at a given point in time.	Blob snapshot <a href="#">Learn more...</a>	Recommended as an alternative to blob versioning when versioning is not appropriate for your scenario, due to cost or other considerations, or when the storage account has a hierarchical namespace enabled.	A blob may be restored from a snapshot if the blob is overwritten. If the blob is deleted, snapshots are also deleted.	Yes, in preview
A blob can be deleted or overwritten, but the data is regularly copied to a second storage account.	Roll-your-own solution for copying data to a second account by using Azure Storage object replication or a tool like AzCopy or Azure Data Factory.	Recommended for peace-of-mind protection against unexpected intentional actions or unpredictable scenarios.  Create the second storage account in the same region as the primary account to avoid incurring egress charges.	Data can be restored from the second storage account if the primary account is compromised in any way.	AzCopy and Azure Data Factory are supported.  Object replication is not supported.

## Data protection by resource type

The following table summarizes the Azure Storage data protection options according to the resources they protect.

DATA PROTECTION OPTION	PROTECTS AN ACCOUNT FROM DELETION	PROTECTS A CONTAINER FROM DELETION	PROTECTS AN OBJECT FROM DELETION	PROTECTS AN OBJECT FROM OVERWRITES
Azure Resource Manager lock	Yes	No <sup>1</sup>	No	No

DATA PROTECTION OPTION	PROTECTS AN ACCOUNT FROM DELETION	PROTECTS A CONTAINER FROM DELETION	PROTECTS AN OBJECT FROM DELETION	PROTECTS AN OBJECT FROM OVERWRITES
Immutability policy on a blob version	Yes <sup>2</sup>	Yes <sup>3</sup>	Yes	Yes <sup>4</sup>
Immutability policy on a container	Yes <sup>5</sup>	Yes	Yes	Yes
Container soft delete	No	Yes	No	No
Blob versioning <sup>6</sup>	No	No	Yes	Yes
Blob soft delete	No	No	Yes	Yes
Point-in-time restore <sup>6</sup>	No	No	Yes	Yes
Blob snapshot	No	No	No	Yes
Roll-your-own solution for copying data to a second account <sup>7</sup>	No	Yes	Yes	Yes

<sup>1</sup> An Azure Resource Manager lock does not protect a container from deletion.

<sup>2</sup> Storage account deletion fails if there is at least one container with version-level immutable storage enabled.

<sup>3</sup> Container deletion fails if at least one blob exists in the container, regardless of whether policy is locked or unlocked.

<sup>4</sup> Overwriting the contents of the current version of the blob creates a new version. An immutability policy protects a version's metadata from being overwritten.

<sup>5</sup> While a legal hold or a locked time-based retention policy is in effect at container scope, the storage account is also protected from deletion.

<sup>6</sup> Not currently supported for Data Lake Storage workloads.

<sup>7</sup> AzCopy and Azure Data Factory are options that are supported for both Blob Storage and Data Lake Storage workloads. Object replication is supported for Blob Storage workloads only.

## Recover deleted or overwritten data

If you should need to recover data that has been overwritten or deleted, how you proceed depends on which data protection options you have enabled and which resource was affected. The following table describes the actions that you can take to recover data.

DELETED OR OVERWRITTEN RESOURCE	POSSIBLE RECOVERY ACTIONS	REQUIREMENTS FOR RECOVERY
Storage account	Attempt to recover the deleted storage account <a href="#">Learn more...</a>	The storage account was originally created with the Azure Resource Manager deployment model and was deleted within the past 14 days. A new storage account with the same name has not been created since the original account was deleted.

DELETED OR OVERWRITTEN RESOURCE	POSSIBLE RECOVERY ACTIONS	REQUIREMENTS FOR RECOVERY
Container	Recover the soft-deleted container and its contents <a href="#">Learn more...</a>	Container soft delete is enabled and the container soft delete retention period has not yet expired.
Containers and blobs	Restore data from a second storage account	All container and blob operations have been effectively replicated to a second storage account.
Blob (any type)	Restore a blob from a previous version <sup>1</sup> <a href="#">Learn more...</a>	Blob versioning is enabled and the blob has one or more previous versions.
Blob (any type)	Recover a soft-deleted blob <a href="#">Learn more...</a>	Blob soft delete is enabled and the soft delete retention interval has not expired.
Blob (any type)	Restore a blob from a snapshot <a href="#">Learn more...</a>	The blob has one or more snapshots.
Set of block blobs	Recover a set of block blobs to their state at an earlier point in time <sup>1</sup> <a href="#">Learn more...</a>	Point-in-time restore is enabled and the restore point is within the retention interval. The storage account has not been compromised or corrupted.
Blob version	Recover a soft-deleted version <sup>1</sup> <a href="#">Learn more...</a>	Blob soft delete is enabled and the soft delete retention interval has not expired.

<sup>1</sup> Not currently supported for Data Lake Storage workloads.

## Summary of cost considerations

The following table summarizes the cost considerations for the various data protection options described in this guide.

DATA PROTECTION OPTION	COST CONSIDERATIONS
Azure Resource Manager lock for a storage account	No charge to configure a lock on a storage account.
Immutability policy on a blob version	No charge to enable version-level immutability on a container. Creating, modifying, or deleting a time-based retention policy or legal hold on a blob version results in a write transaction charge.
Immutability policy on a container	No charge to configure an immutability policy on a container.
Container soft delete	No charge to enable container soft delete for a storage account. Data in a soft-deleted container is billed at same rate as active data until the soft-deleted container is permanently deleted.

DATA PROTECTION OPTION	COST CONSIDERATIONS
Blob versioning	<p>No charge to enable blob versioning for a storage account. After blob versioning is enabled, every write or delete operation on a blob in the account creates a new version, which may lead to increased capacity costs.</p> <p>A blob version is billed based on unique blocks or pages. Costs therefore increase as the base blob diverges from a particular version. Changing a blob or blob version's tier may have a billing impact. For more information, see <a href="#">Pricing and billing</a>.</p> <p>Use lifecycle management to delete older versions as needed to control costs. For more information, see <a href="#">Optimize costs by automating Azure Blob Storage access tiers</a>.</p>
Blob soft delete	<p>No charge to enable blob soft delete for a storage account. Data in a soft-deleted blob is billed at same rate as active data until the soft-deleted blob is permanently deleted.</p>
Point-in-time restore	<p>No charge to enable point-in-time restore for a storage account; however, enabling point-in-time restore also enables blob versioning, soft delete, and change feed, each of which may result in additional charges.</p> <p>You are billed for point-in-time restore when you perform a restore operation. The cost of a restore operation depends on the amount of data being restored. For more information, see <a href="#">Pricing and billing</a>.</p>
Blob snapshots	<p>Data in a snapshot is billed based on unique blocks or pages. Costs therefore increase as the base blob diverges from the snapshot. Changing a blob or snapshot's tier may have a billing impact. For more information, see <a href="#">Pricing and billing</a>.</p> <p>Use lifecycle management to delete older snapshots as needed to control costs. For more information, see <a href="#">Optimize costs by automating Azure Blob Storage access tiers</a>.</p>
Copy data to a second storage account	<p>Maintaining data in a second storage account will incur capacity and transaction costs. If the second storage account is located in a different region than the source account, then copying data to that second account will additionally incur egress charges.</p>

## Disaster recovery

Azure Storage always maintains 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. For more information about how to configure your storage account for high availability, see [Azure Storage redundancy](#).

In the event that a failure occurs in a data center, if your storage account is redundant across two geographical regions (geo-redundant), then you have the option to fail over your account from the primary region to the secondary region. For more information, see [Disaster recovery and storage account failover](#).

Customer-managed failover is not currently supported for storage accounts with a hierarchical namespace

enabled. For more information, see [Blob storage features available in Azure Data Lake Storage Gen2](#).

## Next steps

- [Azure Storage redundancy](#)
- [Disaster recovery and storage account failover](#)

# Soft delete for containers

8/22/2022 • 4 minutes to read • [Edit Online](#)

Container soft delete protects your data from being accidentally deleted by maintaining the deleted data in the system for a specified period of time. During the retention period, you can restore a soft-deleted container and its contents to the container's state at the time it was deleted. After the retention period has expired, the container and its contents are permanently deleted.

## Recommended data protection configuration

Blob soft delete is part of a comprehensive data protection strategy for blob data. For optimal protection for your blob data, Microsoft recommends enabling all of the following data protection features:

- Container soft delete, to restore a container that has been deleted. To learn how to enable container soft delete, see [Enable and manage soft delete for containers](#).
- Blob versioning, to automatically maintain previous versions of a blob. When blob versioning is enabled, you can restore an earlier version of a blob to recover your data if it's erroneously modified or deleted. To learn how to enable blob versioning, see [Enable and manage blob versioning](#).
- Blob soft delete, to restore a blob, snapshot, or version that has been deleted. To learn how to enable blob soft delete, see [Enable and manage soft delete for blobs](#).

To learn more about Microsoft's recommendations for data protection, see [Data protection overview](#).

**Caution**

After you enable blob versioning for a storage account, every write operation to a blob in that account results in the creation of a new version. For this reason, enabling blob versioning may result in additional costs. To minimize costs, use a lifecycle management policy to automatically delete old versions. For more information about lifecycle management, see [Optimize costs by automating Azure Blob Storage access tiers](#).

## How container soft delete works

When you enable container soft delete, you can specify a retention period for deleted containers that is between 1 and 365 days. The default retention period is seven days. During the retention period, you can recover a deleted container by calling the **Restore Container** operation.

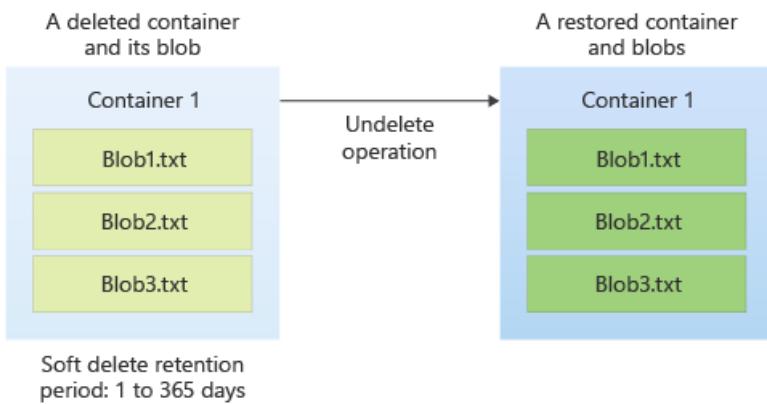
When you restore a container, the container's blobs and any blob versions and snapshots are also restored. However, you can only use container soft delete to restore blobs if the container itself was deleted. To restore a deleted blob when its parent container hasn't been deleted, you must use blob soft delete or blob versioning.

**WARNING**

Container soft delete can restore only whole containers and their contents at the time of deletion. You cannot restore a deleted blob within a container by using container soft delete. Microsoft recommends also enabling blob soft delete and blob versioning to protect individual blobs in a container.

When you restore a container, you must restore it to its original name. If the original name has been used to create a new container, then you will not be able to restore the soft-deleted container.

The following diagram shows how a deleted container can be restored when container soft delete is enabled:



After the retention period has expired, the container is permanently deleted from Azure Storage and can't be recovered. The clock starts on the retention period at the point that the container is deleted. You can change the retention period at any time, but keep in mind that an updated retention period applies only to newly deleted containers. Previously deleted containers will be permanently deleted based on the retention period that was in effect at the time that the container was deleted.

Disabling container soft delete doesn't result in permanent deletion of containers that were previously soft-deleted. Any soft-deleted containers will be permanently deleted at the expiration of the retention period that was in effect at the time that the container was deleted.

Container soft delete is available for the following types of storage accounts:

- General-purpose v2 and v1 storage accounts
- Block blob storage accounts
- Blob storage accounts

Storage accounts with a hierarchical namespace enabled for use with Azure Data Lake Storage Gen2 are also supported.

Version 2019-12-12 or higher of the Azure Storage REST API supports container soft delete.

#### IMPORTANT

Container soft delete does not protect against the deletion of a storage account, but only against the deletion of containers in that account. To protect a storage account from deletion, configure a lock on the storage account resource. For more information about locking Azure Resource Manager resources, see [Lock resources to prevent unexpected changes](#).

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## Pricing and billing

There's no additional charge to enable container soft delete. Data in soft deleted containers is billed at the same rate as active data.

## Next steps

- [Configure container soft delete](#)

- Soft delete for blobs
- Blob versioning

# Soft delete for blobs

8/22/2022 • 12 minutes to read • [Edit Online](#)

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.

## Recommended data protection configuration

Blob soft delete is part of a comprehensive data protection strategy for blob data. For optimal protection for your blob data, Microsoft recommends enabling all of the following data protection features:

- Container soft delete, to restore a container that has been deleted. To learn how to enable container soft delete, see [Enable and manage soft delete for containers](#).
- Blob versioning, to automatically maintain previous versions of a blob. When blob versioning is enabled, you can restore an earlier version of a blob to recover your data if it's erroneously modified or deleted. To learn how to enable blob versioning, see [Enable and manage blob versioning](#).
- Blob soft delete, to restore a blob, snapshot, or version that has been deleted. To learn how to enable blob soft delete, see [Enable and manage soft delete for blobs](#).

To learn more about Microsoft's recommendations for data protection, see [Data protection overview](#).

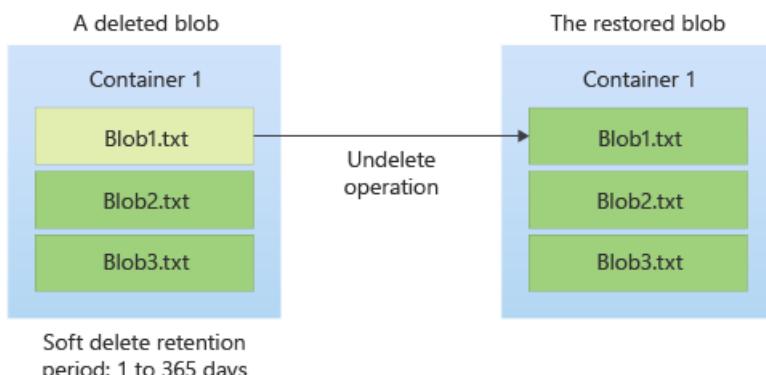
**Caution**

After you enable blob versioning for a storage account, every write operation to a blob in that account results in the creation of a new version. For this reason, enabling blob versioning may result in additional costs. To minimize costs, use a lifecycle management policy to automatically delete old versions. For more information about lifecycle management, see [Optimize costs by automating Azure Blob Storage access tiers](#).

## 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.

Blob versioning is available for general-purpose v2, block blob, and Blob storage accounts. Storage accounts with a hierarchical namespace aren't currently supported.

Version 2017-07-29 and higher of the Azure Storage REST API support blob soft delete.

#### IMPORTANT

You can use blob soft delete only to restore an individual blob, snapshot, directory (in a hierarchical namespace) or version. To restore a container and its contents, container soft delete must also be enabled for the storage account.

Microsoft recommends enabling container soft delete and blob versioning together with blob soft delete to ensure complete protection for blob data. For more information, see [Data protection overview](#).

Blob soft delete does not protect against the deletion of a storage account. To protect a storage account from deletion, configure a lock on the storage account resource. For more information about locking a storage account, see [Apply an Azure Resource Manager lock to a storage account](#).

### How deletions are handled when soft delete is enabled

When blob soft delete is enabled, deleting a blob marks that blob as soft-deleted. No snapshot is created. When the retention period expires, the soft-deleted blob is permanently deleted.

If a blob has snapshots, the blob can't be deleted unless the snapshots are also deleted. When you delete a blob and its snapshots, both the blob and snapshots are marked as soft-deleted. No new snapshots are created.

You can also delete one or more active snapshots without deleting the base blob. In this case, the snapshot is soft-deleted.

If a directory is deleted in an account that has the hierarchical namespace feature enabled on it, the directory and all its contents are marked as soft-deleted. Only the soft-deleted directory can be accessed. In order to access the contents of the soft-deleted directory, the soft-deleted directory needs to be undeleted first.

Soft-deleted objects are invisible unless they're explicitly displayed or listed. For more information about how to list soft-deleted objects, see [Manage and restore soft-deleted blobs](#).

### How overwrites are handled when soft delete is enabled

#### IMPORTANT

This section doesn't apply to accounts that have a hierarchical namespace.

Calling an operation such as [Put Blob](#), [Put Block List](#), or [Copy Blob](#) overwrites the data in a blob. When blob soft delete is enabled, overwriting a blob automatically creates a soft-deleted snapshot of the blob's state prior to the write operation. When the retention period expires, the soft-deleted snapshot is permanently deleted.

Soft-deleted snapshots are invisible unless soft-deleted objects are explicitly displayed or listed. For more information about how to list soft-deleted objects, see [Manage and restore soft-deleted blobs](#).

To protect a copy operation, blob soft delete must be enabled for the destination storage account.

Blob soft delete doesn't protect against operations to write blob metadata or properties. No soft-deleted snapshot is created when a blob's metadata or properties are updated.

Blob soft delete doesn't afford overwrite protection for blobs in the archive tier. If a blob in the archive tier is

overwritten with a new blob in any tier, then the overwritten blob is permanently deleted.

For premium storage accounts, soft-deleted snapshots don't count toward the per-blob limit of 100 snapshots.

### Restoring soft-deleted objects

You can restore soft-deleted blobs or directories (in a hierarchical namespace) by calling the [Undelete Blob](#) operation within the retention period. The **Undelete Blob** operation restores a blob and any soft-deleted snapshots associated with it. Any snapshots that were deleted during the retention period are restored.

In accounts that have a hierarchical namespace, the **Undelete Blob** operation can also be used to restore a soft-deleted directory and all its contents. If you rename a directory that contains soft deleted blobs, those soft deleted blobs become disconnected from the directory. If you want to restore those blobs, you'll have to revert the name of the directory back to its original name or create a separate directory that uses the original directory name. Otherwise, you'll receive an error when you attempt to restore those soft deleted blobs. You also cannot restore a directory or a blob to a filepath that has a directory or blob of that name already there. For example, if you delete a.txt (1) and upload a new file also named a.txt (2), you cannot restore the soft deleted a.txt (1) until the active a.txt (2) has either been deleted or renamed. You cannot access the contents of a soft-deleted directory until after the directory has been undeleted.

Calling **Undelete Blob** on a blob that isn't soft-deleted will restore any soft-deleted snapshots that are associated with the blob. If the blob has no snapshots and isn't soft-deleted, then calling **Undelete Blob** has no effect.

To promote a soft-deleted snapshot to the base blob, first call **Undelete Blob** on the base blob to restore the blob and its snapshots. Next, copy the desired snapshot over the base blob. You can also copy the snapshot to a new blob.

Data in a soft-deleted blob or snapshot can't be read until the object has been restored.

For more information on how to restore soft-deleted objects, see [Manage and restore soft-deleted blobs](#).

## Blob soft delete and versioning

### IMPORTANT

Versioning is not supported for accounts that have a hierarchical namespace.

If blob versioning and blob soft delete are both enabled for a storage account, then overwriting a blob automatically creates a new previous version that reflects the blob's state before the write operation. The new version isn't soft-deleted and isn't removed when the soft-delete retention period expires. No soft-deleted snapshots are created.

If blob versioning and blob soft delete are both enabled for a storage account, then when you delete a blob, the current version of the blob becomes a previous version, and there's no longer a current version. No new version is created and no soft-deleted snapshots are created. All previous versions are retained until they are explicitly deleted, either with a direct delete operation or via a lifecycle management policy.

Enabling soft delete and versioning together protects previous blob versions as well as current versions from deletion. When soft delete is enabled, explicitly deleting a previous version creates a soft-deleted version that is retained until the soft-delete retention period elapses. After the soft-delete retention period has elapsed, the soft-deleted blob version is permanently deleted.

You can use the **Undelete Blob** operation to restore soft-deleted versions during the soft-delete retention period. The **Undelete Blob** operation always restores all soft-deleted versions of the blob. It isn't possible to restore only a single soft-deleted version.

**NOTE**

Calling the **Undelete Blob** operation on a deleted blob when versioning is enabled restores any soft-deleted versions or snapshots, but does not restore the current version. To restore the current version, promote a previous version by copying it to the current version.

Microsoft recommends enabling both versioning and blob soft delete for your storage accounts for optimal data protection. For more information about using blob versioning and soft delete together, see [Blob versioning and soft delete](#).

## Blob soft delete protection by operation

The following table describes the expected behavior for delete and write operations when blob soft delete is enabled, either with or without blob versioning.

### Storage account (no hierarchical namespace)

REST API OPERATIONS	SOFT DELETE ENABLED	SOFT DELETE AND VERSIONING ENABLED
<a href="#">Delete Storage Account</a>	No change. Containers and blobs in the deleted account aren't recoverable.	No change. Containers and blobs in the deleted account aren't recoverable.
<a href="#">Delete Container</a>	No change. Blobs in the deleted container aren't recoverable.	No change. Blobs in the deleted container aren't recoverable.
<a href="#">Delete Blob</a>	If used to delete a blob, that blob is marked as soft deleted.  If used to delete a blob snapshot, the snapshot is marked as soft deleted.	If used to delete a blob, the current version becomes a previous version, and the current version is deleted. No new version is created and no soft-deleted snapshots are created.  If used to delete a blob version, the version is marked as soft deleted.
<a href="#">Undelete Blob</a>	Restores a blob and any snapshots that were deleted within the retention period.	Restores a blob and any versions that were deleted within the retention period.
<a href="#">Put Blob</a> <a href="#">Put Block List</a> <a href="#">Copy Blob</a> <a href="#">Copy Blob from URL</a>	If called on an active blob, then a snapshot of the blob's state prior to the operation is automatically generated.  If called on a soft-deleted blob, then a snapshot of the blob's prior state is generated only if it's being replaced by a blob of the same type. If the blob is of a different type, then all existing soft deleted data is permanently deleted.	A new version that captures the blob's state prior to the operation is automatically generated.

REST API OPERATIONS	SOFT DELETE ENABLED	SOFT DELETE AND VERSIONING ENABLED
Put Block  <a href="#">Put Block from URL</a>	If used to commit a block to an active blob, there's no change.  If used to commit a block to a blob that is soft-deleted, a new blob is created and a snapshot is automatically generated to capture the state of the soft-deleted blob.	No change.
Put Page  <a href="#">Put Page from URL</a>	No change. Page blob data that is overwritten or cleared using this operation isn't saved and isn't recoverable.	No change. Page blob data that is overwritten or cleared using this operation isn't saved and isn't recoverable.
Append Block  <a href="#">Append Block from URL</a>	No change.	No change.
<a href="#">Set Blob Properties</a>	No change. Overwritten blob properties aren't recoverable.	No change. Overwritten blob properties aren't recoverable.
<a href="#">Set Blob Metadata</a>	No change. Overwritten blob metadata isn't recoverable.	A new version that captures the blob's state prior to the operation is automatically generated.
<a href="#">Set Blob Tier</a>	The base blob is moved to the new tier. Any active or soft-deleted snapshots remain in the original tier. No soft-deleted snapshot is created.	The base blob is moved to the new tier. Any active or soft-deleted versions remain in the original tier. No new version is created.

### Storage account (hierarchical namespace)

REST API OPERATION	SOFT DELETE ENABLED
<a href="#">Path - Delete</a>	A soft deleted blob or directory is created. The soft deleted object is deleted after the retention period.
<a href="#">Delete Blob</a>	A soft deleted object is created. The soft deleted object is deleted after the retention period. Soft delete won't be supported for blobs with snapshots and snapshots.
<a href="#">Path - Create</a> that renames a blob or directory	Existing destination blob or empty directory will get soft deleted and the source will replace it. The soft deleted object is deleted after the retention period.

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## Pricing and billing

All soft deleted data is billed at the same rate as active data. You won't be charged for data that is permanently deleted after the retention period elapses.

When you enable soft delete, Microsoft recommends using a short retention period to better understand how the feature will affect your bill. The minimum recommended retention period is seven days.

Enabling soft delete for frequently overwritten data may result in increased storage capacity charges and increased latency when listing blobs. You can mitigate this additional cost and latency by storing the frequently overwritten data in a separate storage account where soft delete is disabled.

You aren't billed for transactions related to the automatic generation of snapshots or versions when a blob is overwritten or deleted. You're billed for calls to the **Undelete Blob** operation at the transaction rate for write operations.

For more information on pricing for Blob Storage, see the [Blob Storage pricing](#) page.

## Blob soft delete and virtual machine disks

Blob soft delete is available for both premium and standard unmanaged disks, which are page blobs under the covers. Soft delete can help you recover data deleted or overwritten by the [Delete Blob](#), [Put Blob](#), [Put Block List](#), and [Copy Blob](#) operations only.

Data that is overwritten by a call to [Put Page](#) isn't recoverable. An Azure virtual machine writes to an unmanaged disk using calls to [Put Page](#), so using soft delete to undo writes to an unmanaged disk from an Azure VM isn't a supported scenario.

## Next steps

- [Enable soft delete for blobs](#)
- [Manage and restore soft-deleted blobs](#)
- [Blob versioning](#)

# Blob versioning

8/22/2022 • 17 minutes to read • [Edit Online](#)

You can enable Blob storage versioning to automatically maintain previous versions of an object. When blob versioning is enabled, you can access earlier versions of a blob to recover your data if it is modified or deleted.

## Recommended data protection configuration

Blob versioning is part of a comprehensive data protection strategy for blob data. For optimal protection for your blob data, Microsoft recommends enabling all of the following data protection features:

- Blob versioning, to automatically maintain previous versions of a blob. When blob versioning is enabled, you can restore an earlier version of a blob to recover your data if it is erroneously modified or deleted. To learn how to enable blob versioning, see [Enable and manage blob versioning](#).
- Container soft delete, to restore a container that has been deleted. To learn how to enable container soft delete, see [Enable and manage soft delete for containers](#).
- Blob soft delete, to restore a blob, snapshot, or version that has been deleted. To learn how to enable blob soft delete, see [Enable and manage soft delete for blobs](#).

To learn more about Microsoft's recommendations for data protection, see [Data protection overview](#).

**Caution**

After you enable blob versioning for a storage account, every write operation to a blob in that account results in the creation of a new version. For this reason, enabling blob versioning may result in additional costs. To minimize costs, use a lifecycle management policy to automatically delete old versions. For more information about lifecycle management, see [Optimize costs by automating Azure Blob Storage access tiers](#).

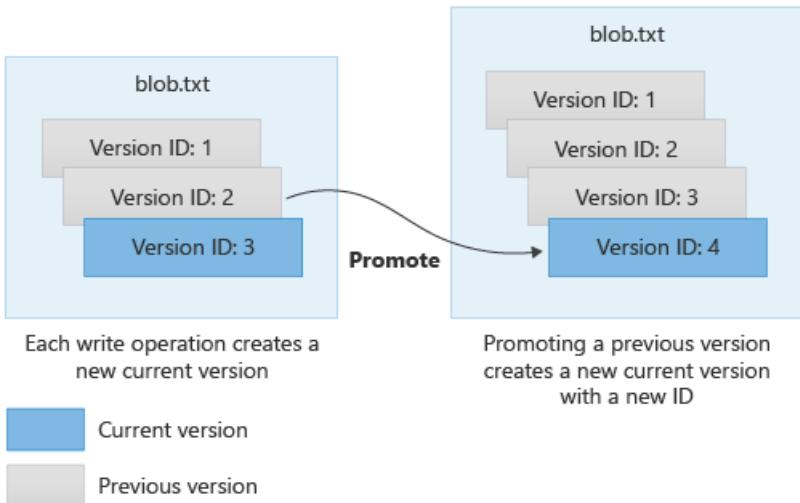
## How blob versioning works

A version captures the state of a blob at a given point in time. Each version is identified with a version ID. When blob versioning is enabled for a storage account, Azure Storage automatically creates a new version with a unique ID when a blob is first created and each time that the blob is subsequently modified.

A version ID can identify the current version or a previous version. A blob can have only one current version at a time.

When you create a new blob, a single version exists, and that version is the current version. When you modify an existing blob, the current version becomes a previous version. A new version is created to capture the updated state, and that new version is the current version. When you delete a blob, the current version of the blob becomes a previous version, and there is no longer a current version. Any previous versions of the blob persist.

The following diagram shows how versions are created on write operations, and how a previous version may be promoted to be the current version:



Blob versions are immutable. You cannot modify the content or metadata of an existing blob version.

Having a large number of versions per blob can increase the latency for blob listing operations. Microsoft recommends maintaining fewer than 1000 versions per blob. You can use lifecycle management to automatically delete old versions. For more information about lifecycle management, see [Optimize costs by automating Azure Blob Storage access tiers](#).

Blob versioning is available for standard general-purpose v2, premium block blob, and legacy Blob storage accounts. Storage accounts with a hierarchical namespace enabled for use with Azure Data Lake Storage Gen2 are not currently supported.

Version 2019-10-10 and higher of the Azure Storage REST API supports blob versioning.

#### IMPORTANT

Blob versioning cannot help you to recover from the accidental deletion of a storage account or container. To prevent accidental deletion of the storage account, configure a lock on the storage account resource. For more information on locking a storage account, see [Apply an Azure Resource Manager lock to a storage account](#).

## Version ID

Each blob version is identified by a unique version ID. The value of the version ID is the timestamp at which the blob was updated. The version ID is assigned at the time that the version is created.

You can perform read or delete operations on a specific version of a blob by providing its version ID. If you omit the version ID, the operation acts against the current version.

When you call a write operation to create or modify a blob, Azure Storage returns the *x-ms-version-id* header in the response. This header contains the version ID for the current version of the blob that was created by the write operation.

The version ID remains the same for the lifetime of the version.

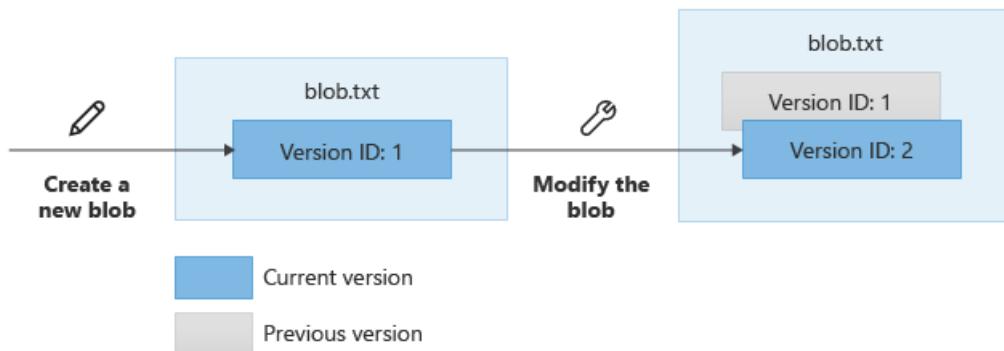
### Versioning on write operations

When blob versioning is turned on, each write operation to a blob creates a new version. Write operations include [Put Blob](#), [Put Block List](#), [Copy Blob](#), and [Set Blob Metadata](#).

If the write operation creates a new blob, then the resulting blob is the current version of the blob. If the write operation modifies an existing blob, then the current version becomes a previous version, and a new current version is created to capture the updated blob.

The following diagram shows how write operations affect blob versions. For simplicity, the diagrams shown in this article display the version ID as a simple integer value. In reality, the version ID is a timestamp. The current

version is shown in blue, and previous versions are shown in gray.



#### NOTE

A blob that was created prior to versioning being enabled for the storage account does not have a version ID. When that blob is modified, the modified blob becomes the current version, and a version is created to save the blob's state before the update. The version is assigned a version ID that is its creation time.

When blob versioning is enabled for a storage account, all write operations on block blobs trigger the creation of a new version, with the exception of the [Put Block](#) operation.

For page blobs and append blobs, only a subset of write operations trigger the creation of a version. These operations include:

- [Put Blob](#)
- [Put Block List](#)
- [Set Blob Metadata](#)
- [Copy Blob](#)

The following operations do not trigger the creation of a new version. To capture changes from those operations, take a manual snapshot:

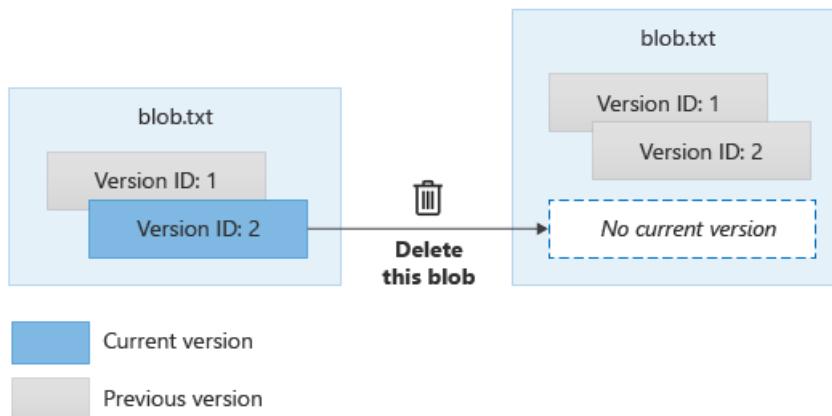
- [Put Page](#) (page blob)
- [Append Block](#) (append blob)

All versions of a blob must be of the same blob type. If a blob has previous versions, you cannot overwrite a blob of one type with another type unless you first delete the blob and all of its versions.

#### Versioning on delete operations

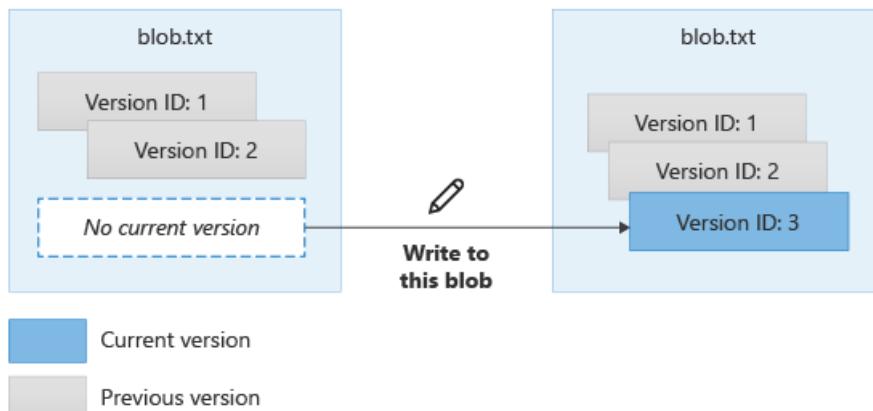
When you call the [Delete Blob](#) operation without specifying a version ID, the current version becomes a previous version, and there is no longer a current version. All existing previous versions of the blob are preserved.

The following diagram shows the effect of a delete operation on a versioned blob:



To delete a specific version of a blob, provide the ID for that version on the delete operation. If blob soft delete is also enabled for the storage account, the version is maintained in the system until the soft delete retention period elapses.

Writing new data to the blob creates a new current version of the blob. Any existing versions are unaffected, as shown in the following diagram.



## Access tiers

You can move any version of a block blob, including the current version, to a different blob access tier by calling the [Set Blob Tier](#) operation. You can take advantage of lower capacity pricing by moving older versions of a blob to the cool or archive tier. For more information, see [Hot, Cool, and Archive access tiers for blob data](#).

To automate the process of moving block blobs to the appropriate tier, use blob life cycle management. For more information on life cycle management, see [Manage the Azure Blob storage life cycle](#).

## Enable or disable blob versioning

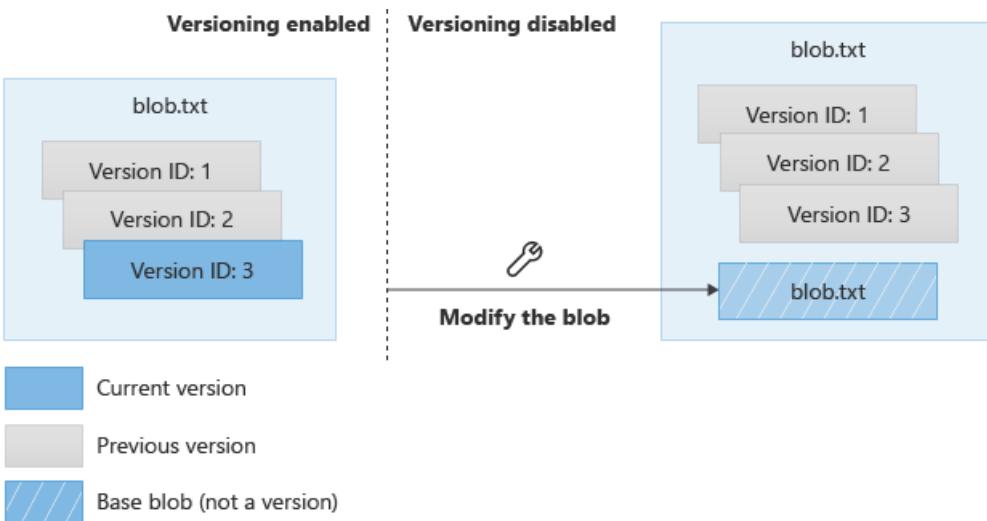
To learn how to enable or disable blob versioning, see [Enable and manage blob versioning](#).

Disabling blob versioning does not delete existing blobs, versions, or snapshots. When you turn off blob versioning, any existing versions remain accessible in your storage account. No new versions are subsequently created.

After versioning is disabled, modifying the current version creates a blob that is not a version. All subsequent updates to the blob will overwrite its data without saving the previous state. All existing versions persist as previous versions.

You can read or delete versions using the version ID after versioning is disabled. You can also list a blob's versions after versioning is disabled.

The following diagram shows how modifying a blob after versioning is disabled creates a blob that is not versioned. Any existing versions associated with the blob persist.



## Blob versioning and soft delete

Blob versioning and blob soft delete are part of the recommended data protection configuration for storage accounts. For more information about Microsoft's recommendations for data protection, see [Recommended data protection configuration](#) in this article, as well as [Data protection overview](#).

### Overwriting a blob

If blob versioning and blob soft delete are both enabled for a storage account, then overwriting a blob automatically creates a new version. The new version is not soft-deleted and is not removed when the soft-delete retention period expires. No soft-deleted snapshots are created.

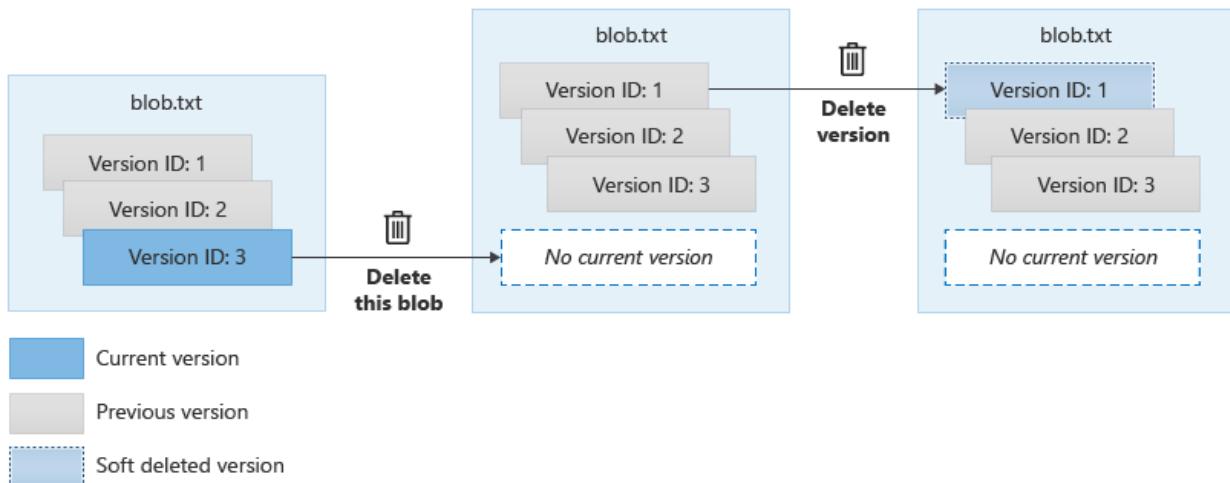
### Deleting a blob or version

If versioning and soft delete are both enabled for a storage account, then when you delete a blob, the current version of the blob becomes a previous version. No new version is created and no soft-deleted snapshots are created. The soft delete retention period is not in effect for the deleted blob.

Soft delete offers additional protection for deleting blob versions. When you delete a previous version of the blob, that version is soft-deleted. The soft-deleted version is preserved until the soft delete retention period elapses, at which point it is permanently deleted.

To delete a previous version of a blob, call the [Delete Blob](#) operation and specify the version ID.

The following diagram shows what happens when you delete a blob or a blob version.



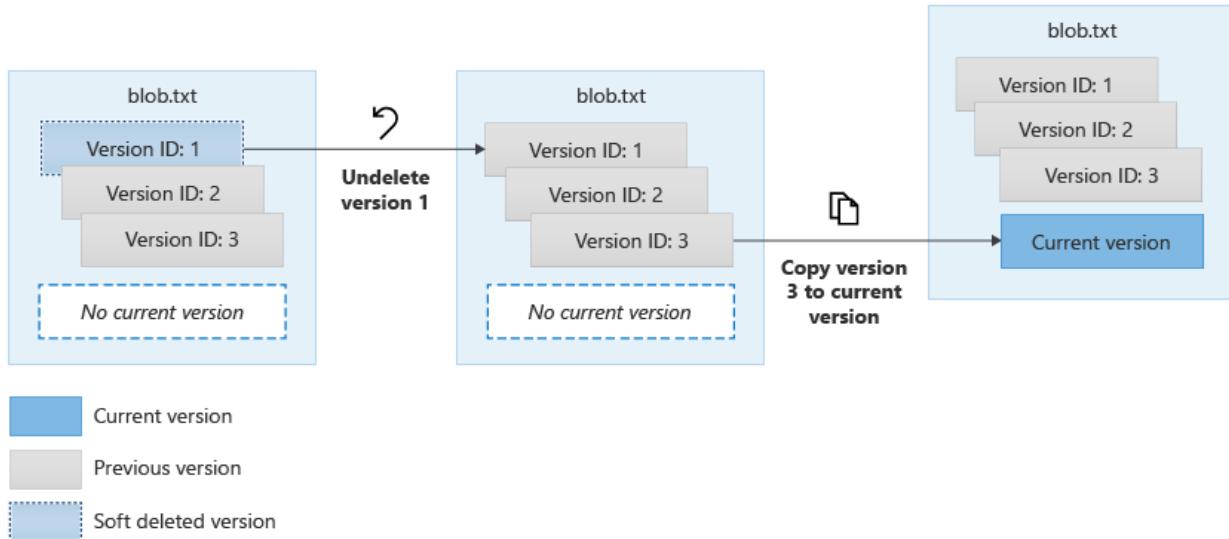
### Restoring a soft-deleted version

You can use the [Undelete Blob](#) operation to restore soft-deleted versions during the soft delete retention period. The [Undelete Blob](#) operation always restores all soft-deleted versions of the blob. It is not possible to restore

only a single soft-deleted version.

Restoring soft-deleted versions with the **Undelete Blob** operation does not promote any version to be the current version. To restore the current version, first restore all soft-deleted versions, and then use the [Copy Blob](#) operation to copy a previous version to a new current version.

The following diagram shows how to restore soft-deleted blob versions with the **Undelete Blob** operation, and how to restore the current version of the blob with the **Copy Blob** operation.



After the soft-delete retention period has elapsed, any soft-deleted blob versions are permanently deleted.

## Blob versioning and blob snapshots

A blob snapshot is a read-only copy of a blob that's taken at a specific point in time. Blob snapshots and blob versions are similar, but a snapshot is created manually by you or your application, while a blob version is created automatically on a write or delete operation when blob versioning is enabled for your storage account.

### IMPORTANT

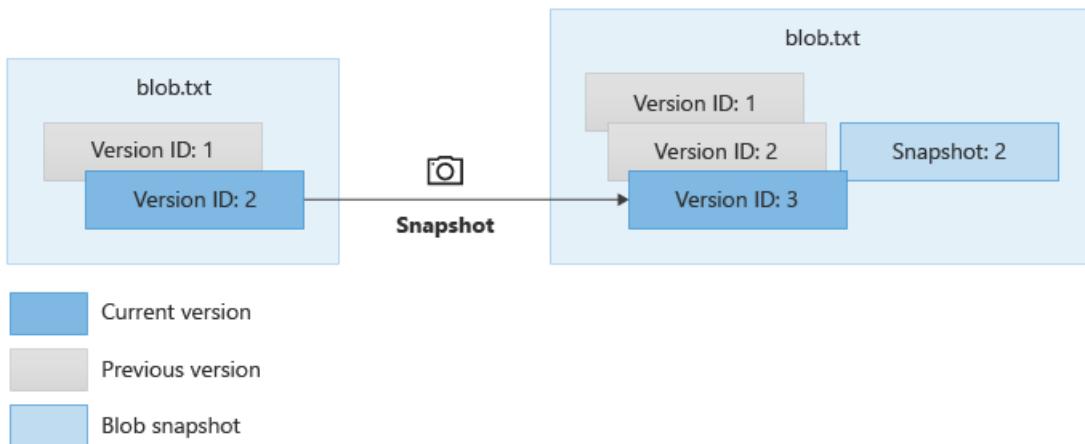
Microsoft recommends that after you enable blob versioning, you also update your application to stop taking snapshots of block blobs. If versioning is enabled for your storage account, all block blob updates and deletions are captured and preserved by versions. Taking snapshots does not offer any additional protections to your block blob data if blob versioning is enabled, and may increase costs and application complexity.

### Snapshot a blob when versioning is enabled

Although it is not recommended, you can take a snapshot of a blob that is also versioned. If you cannot update your application to stop taking snapshots of blobs when you enable versioning, your application can support both snapshots and versions.

When you take a snapshot of a versioned blob, a new version is created at the same time that the snapshot is created. A new current version is also created when a snapshot is taken.

The following diagram shows what happens when you take a snapshot of a versioned blob. In the diagram, blob versions and snapshots with version ID 2 and 3 contain identical data.



## Authorize operations on blob versions

You can authorize access to blob versions using one of the following approaches:

- By using Azure role-based access control (Azure RBAC) to grant permissions to an Azure Active Directory (Azure AD) security principal. Microsoft recommends using Azure AD for superior security and ease of use. For more information about using Azure AD with blob operations, see [Authorize access to data in Azure Storage](#).
- By using a shared access signature (SAS) to delegate access to blob versions. Specify the version ID for the signed resource type `bv`, representing a blob version, to create a SAS token for operations on a specific version. For more information about shared access signatures, see [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#).
- By using the account access keys to authorize operations against blob versions with Shared Key. For more information, see [Authorize with Shared Key](#).

Blob versioning is designed to protect your data from accidental or malicious deletion. To enhance protection, deleting a blob version requires special permissions. The following sections describe the permissions needed to delete a blob version.

### Azure RBAC action to delete a blob version

The following table shows which Azure RBAC actions support deleting a blob or a blob version.

DESCRIPTION	BLOB SERVICE OPERATION	AZURE RBAC DATA ACTION REQUIRED	AZURE BUILT-IN ROLE SUPPORT
Deleting the current version	Delete Blob	<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/delete</code>	Storage Blob Data Contributor
Deleting a previous version	Delete Blob	<code>Microsoft.Storage/storageAccounts/blobServices/containers/blobs/deleteBlobVersion/action</code>	Storage Blob Data Owner

### Shared access signature (SAS) parameters

The signed resource for a blob version is `bv`. For more information, see [Create a service SAS](#) or [Create a user delegation SAS](#).

The following table shows the permission required on a SAS to delete a blob version.

PERMISSION	URI SYMBOL	ALLOWED OPERATIONS
Delete	x	Delete a blob version.

## Pricing and billing

Enabling blob versioning can result in additional data storage charges to your account. When designing your application, it is important to be aware of how these charges might accrue so that you can minimize costs.

Blob versions, like blob snapshots, are billed at the same rate as active data. How versions are billed depends on whether you have explicitly set the tier for the current or previous versions of a blob (or snapshots). For more information about blob tiers, see [Hot, Cool, and Archive access tiers for blob data](#).

If you have not changed a blob or version's tier, then you are billed for unique blocks of data across that blob, its versions, and any snapshots it may have. For more information, see [Billing when the blob tier has not been explicitly set](#).

If you have changed a blob or version's tier, then you are billed for the entire object, regardless of whether the blob and version are eventually in the same tier again. For more information, see [Billing when the blob tier has been explicitly set](#).

### NOTE

Enabling versioning for data that is frequently overwritten may result in increased storage capacity charges and increased latency during listing operations. To mitigate these concerns, store frequently overwritten data in a separate storage account with versioning disabled.

For more information about billing details for blob snapshots, see [Blob snapshots](#).

### Billing when the blob tier has not been explicitly set

If you have not explicitly set the blob tier for any versions of a blob, then you are charged for unique blocks or pages across all versions, and any snapshots it may have. Data that is shared across blob versions is charged only once. When a blob is updated, then data in the new current version diverges from the data stored in previous versions, and the unique data is charged per block or page.

When you replace a block within a block blob, that block is subsequently charged as a unique block. This is true even if the block has the same block ID and the same data as it has in the previous version. After the block is committed again, it diverges from its counterpart in the previous version, and you will be charged for its data. The same holds true for a page in a page blob that's updated with identical data.

Blob storage does not have a means to determine whether two blocks contain identical data. Each block that is uploaded and committed is treated as unique, even if it has the same data and the same block ID. Because charges accrue for unique blocks, it's important to keep in mind that updating a blob when versioning is enabled will result in additional unique blocks and additional charges.

When blob versioning is enabled, call update operations on block blobs so that they update the least possible number of blocks. The write operations that permit fine-grained control over blocks are [Put Block](#) and [Put Block List](#). The [Put Blob](#) operation, on the other hand, replaces the entire contents of a blob and so may lead to additional charges.

The following scenarios demonstrate how charges accrue for a block blob and its versions when the blob tier has not been explicitly set.

#### Scenario 1

In scenario 1, the blob has a previous version. The blob has not been updated since the version was created, so

charges are incurred only for unique blocks 1, 2, and 3.

Base Blob		Version	
ID = 1	AAA	ID = 1	AAA
ID = 2	BBB	ID = 2	BBB
ID = 3	CCC	ID = 3	CCC

#### Scenario 2

In scenario 2, one block (block 3 in the diagram) in the blob has been updated. Even though the updated block contains the same data and the same ID, it is not the same as block 3 in the previous version. As a result, the account is charged for four blocks.

Base Blob		Version	
ID = 1	AAA	ID = 1	AAA
ID = 2	BBB	ID = 2	BBB
ID = 3	CCC	ID = 3	CCC

#### Scenario 3

In scenario 3, the blob has been updated, but the version has not. Block 3 was replaced with block 4 in the current blob, but the previous version still reflects block 3. As a result, the account is charged for four blocks.

Base Blob		Version	
ID = 1	AAA	ID = 1	AAA
ID = 2	BBB	ID = 2	BBB
ID = 4	DDD	ID = 3	CCC

#### Scenario 4

In scenario 4, the current version has been completely updated and contains none of its original blocks. As a result, the account is charged for all eight unique blocks — four in the current version, and four combined in the two previous versions. This scenario can occur if you are writing to a blob with the [Put Blob](#) operation, because it replaces the entire contents of the blob.

Base Blob		Version 1		Version 2	
ID = 5	EEE	ID = 1	AAA	ID = 1	AAA
ID = 6	FFF	ID = 2	BBB	ID = 2	BBB
ID = 7	GGG	ID = 3	CCC	ID = 4	DDD
ID = 8	HHH				

#### Billing when the blob tier has been explicitly set

If you have explicitly set the blob tier for a blob or version (or snapshot), then you are charged for the full content length of the object in the new tier, regardless of whether it shares blocks with an object in the original tier. You are also charged for the full content length of the oldest version in the original tier. Any other previous versions or snapshots that remain in the original tier are charged for unique blocks that they may share, as described in [Billing when the blob tier has not been explicitly set](#).

#### Moving a blob to a new tier

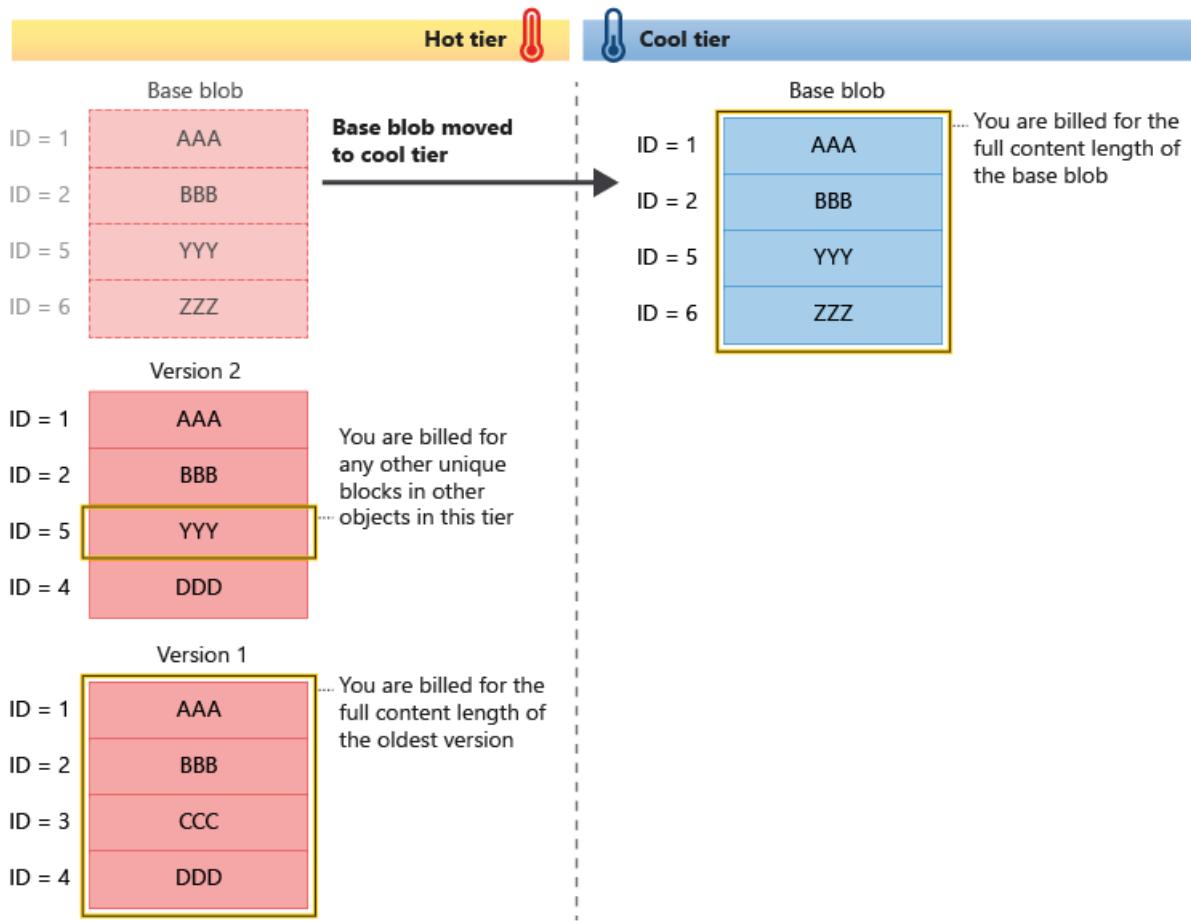
The following table describes the billing behavior for a blob or version when it is moved to a new tier.

WHEN BLOB TIER IS SET...	THEN YOU ARE BILLED FOR...
Explicitly on a version, whether current or previous	The full content length of that version. Versions that don't have an explicitly set tier are billed only for unique blocks. <sup>1</sup>
To archive	The full content length of all versions and snapshots. <sup>1</sup>

<sup>1</sup>If there are other previous versions or snapshots that have not been moved from their original tier, those versions or snapshots are charged based on the number of unique blocks they contain, as described in [Billing](#)

when the blob tier has not been explicitly set.

The following diagram illustrates how objects are billed when a versioned blob is moved to a different tier.



Explicitly setting the tier for a blob, version, or snapshot cannot be undone. If you move a blob to a new tier and then move it back to its original tier, you are charged for the full content length of the object even if it shares blocks with other objects in the original tier.

Operations that explicitly set the tier of a blob, version, or snapshot include:

- [Set Blob Tier](#)
- [Put Blob](#) with tier specified
- [Put Block List](#) with tier specified
- [Copy Blob](#) with tier specified

#### Deleting a blob when soft delete is enabled

When blob soft delete is enabled, all soft-deleted entities are billed at full content length. If you delete or overwrite a current version that has had its tier explicitly set, then any previous versions of the soft-deleted blob are billed at full content length. For more information about how blob versioning and soft delete work together, see [Blob versioning and soft delete](#).

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## See also

- [Enable and manage blob versioning](#)
- [Creating a snapshot of a blob](#)
- [Soft delete for blobs](#)
- [Soft delete for containers](#)

# Point-in-time restore for block blobs

8/22/2022 • 8 minutes to read • [Edit Online](#)

Point-in-time restore provides protection against accidental deletion or corruption by enabling you to restore block blob data to an earlier state. Point-in-time restore is useful in scenarios where a user or application accidentally deletes data or where an application error corrupts data. Point-in-time restore also enables testing scenarios that require reverting a data set to a known state before running further tests.

Point-in-time restore is supported for general-purpose v2 storage accounts in the standard performance tier only. Only data in the hot and cool access tiers can be restored with point-in-time restore.

To learn how to enable point-in-time restore for a storage account, see [Perform a point-in-time restore on block blob data](#).

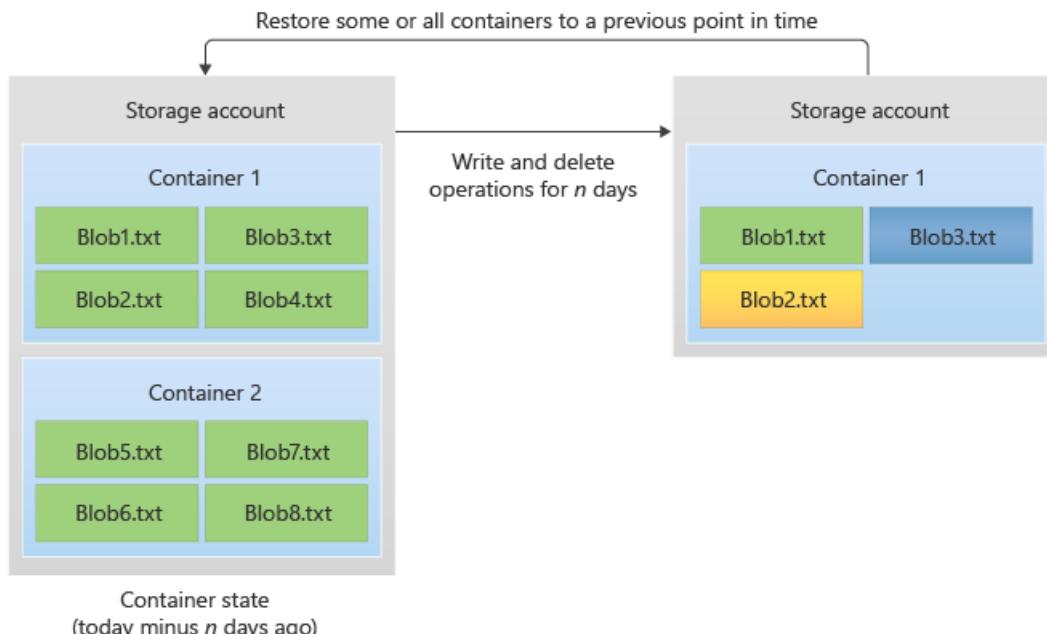
## How point-in-time restore works

To enable point-in-time restore, you create a management policy for the storage account and specify a retention period. During the retention period, you can restore block blobs from the present state to a state at a previous point in time.

To initiate a point-in-time restore, call the [Restore Blob Ranges](#) operation and specify a restore point in UTC time. You can specify lexicographical ranges of container and blob names to restore, or omit the range to restore all containers in the storage account. Up to 10 lexicographical ranges are supported per restore operation.

Azure Storage analyzes all changes that have been made to the specified blobs between the requested restore point, specified in UTC time, and the present moment. The restore operation is atomic, so it either succeeds completely in restoring all changes, or it fails. If there are any blobs that cannot be restored, then the operation fails, and read and write operations to the affected containers resume.

The following diagram shows how point-in-time restore works. One or more containers or blob ranges is restored to its state  $n$  days ago, where  $n$  is less than or equal to the retention period defined for point-in-time restore. The effect is to revert write and delete operations that happened during the retention period.



Only one restore operation can be run on a storage account at a time. A restore operation cannot be canceled

once it is in progress, but a second restore operation can be performed to undo the first operation.

The **Restore Blob Ranges** operation returns a restore ID that uniquely identifies the operation. To check the status of a point-in-time restore, call the **Get Restore Status** operation with the restore ID returned from the **Restore Blob Ranges** operation.

#### IMPORTANT

When you perform a restore operation, Azure Storage blocks data operations on the blobs in the ranges being restored for the duration of the operation. Read, write, and delete operations are blocked in the primary location. For this reason, operations such as listing containers in the Azure portal may not perform as expected while the restore operation is underway.

Read operations from the secondary location may proceed during the restore operation if the storage account is geo-replicated.

#### Caution

Point-in-time restore supports restoring against operations that acted on block blobs only. Any operations that acted on containers cannot be restored. For example, if you delete a container from the storage account by calling the [Delete Container](#) operation, that container cannot be restored with a point-in-time restore operation. Rather than deleting an entire container, delete individual blobs if you may want to restore them later.

#### Prerequisites for point-in-time restore

Point-in-time restore requires that the following Azure Storage features be enabled before you can enable point-in-time restore:

- [Soft delete](#)
- [Change feed](#)
- [Blob versioning](#)

To learn more about Microsoft's recommendations for data protection, see [Data protection overview](#).

#### Caution

After you enable blob versioning for a storage account, every write operation to a blob in that account results in the creation of a new version. For this reason, enabling blob versioning may result in additional costs. To minimize costs, use a lifecycle management policy to automatically delete old versions. For more information about lifecycle management, see [Optimize costs by automating Azure Blob Storage access tiers](#).

#### Retention period for point-in-time restore

When you enable point-in-time restore for a storage account, you specify a retention period. Block blobs in your storage account can be restored during the retention period.

The retention period begins a few minutes after you enable point-in-time restore. Keep in mind that you cannot restore blobs to a state prior to the beginning of the retention period. For example, if you enabled point-in-time restore on May 1st with a retention of 30 days, then on May 15th you can restore to a maximum of 15 days. On June 1st, you can restore data from between 1 and 30 days.

The retention period for point-in-time restore must be at least one day less than the retention period specified for soft delete. For example, if the soft delete retention period is set to 7 days, then the point-in-time restore retention period may be between 1 and 6 days.

#### NOTE

The retention period that you specify for point-in-time restore has no effect on the retention of blob versions. Blob versions are retained until they are explicitly deleted. To optimize costs by deleting or tiering older versions, create a lifecycle management policy. For more information, see [Optimize costs by automatically managing the data lifecycle](#).

The time that it takes to restore a set of data is based on the number of write and delete operations made during the restore period. For example, an account with one million blobs with 3,000 blobs added per day and 1,000 blobs deleted per day will require approximately two hours to restore to a point 30 days in the past. A retention period and restoration more than 90 days in the past would not be recommended for an account with this rate of change.

### Permissions for point-in-time restore

To initiate a restore operation, a client must have write permissions to all containers in the storage account. To grant permissions to authorize a restore operation with Azure Active Directory (Azure AD), assign the **Storage Account Contributor** role to the security principal at the level of the storage account, resource group, or subscription.

## Limitations and known issues

Point-in-time restore for block blobs has the following limitations and known issues:

- Only block blobs in a standard general-purpose v2 storage account can be restored as part of a point-in-time restore operation. Append blobs, page blobs, and premium block blobs are not restored.
- If you have deleted a container during the retention period, that container will not be restored with the point-in-time restore operation. If you attempt to restore a range of blobs that includes blobs in a deleted container, the point-in-time restore operation will fail. To learn about protecting containers from deletion, see [Soft delete for containers](#).
- If you use permanent delete to purge soft-deleted versions of a blob during the point-in-time restore retention period, then a restore operation may not be able to restore that blob correctly.
- If a blob has moved between the hot and cool tiers in the period between the present moment and the restore point, the blob is restored to its previous tier. Restoring block blobs in the archive tier is not supported. For example, if a blob in the hot tier was moved to the archive tier two days ago, and a restore operation restores to a point three days ago, the blob is not restored to the hot tier. To restore an archived blob, first move it out of the archive tier. For more information, see [Overview of blob rehydration from the archive tier](#).
- If an immutability policy is configured, then a restore operation can be initiated, but any blobs that are protected by the immutability policy will not be modified. A restore operation in this case will not result in the restoration of a consistent state to the date and time given.
- A block that has been uploaded via [Put Block](#) or [Put Block from URL](#), but not committed via [Put Block List](#), is not part of a blob and so is not restored as part of a restore operation.
- A blob with an active lease cannot be restored. If a blob with an active lease is included in the range of blobs to restore, the restore operation will fail atomically. Break any active leases prior to initiating the restore operation.
- Performing a customer-managed failover on a storage account resets the earliest possible restore point for that storage account. For example, suppose you have set the retention period to 30 days. If more than 30 days have elapsed since the failover, then you can restore to any point within that 30 days. However, if fewer than 30 days have elapsed since the failover, then you cannot restore to a point prior to the failover, regardless of the retention period. For example, if it's been 10 days since the failover, then the earliest possible restore point is 10 days in the past, not 30 days in the past.
- Snapshots are not created or deleted as part of a restore operation. Only the base blob is restored to its previous state.
- Point-in-time restore is not supported for hierarchical namespaces or operations via Azure Data Lake Storage Gen2.

**IMPORTANT**

If you restore block blobs to a point that is earlier than September 22, 2020, preview limitations for point-in-time restore will be in effect. Microsoft recommends that you choose a restore point that is equal to or later than September 22, 2020 to take advantage of the generally available point-in-time restore feature.

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## Pricing and billing

There is no charge to enable point-in-time restore. However, enabling point-in-time restore also enables blob versioning, soft delete, and change feed, each of which may result in additional charges.

Billing for performing point-in-time restores is based on the amount of changefeed data processed for the restore. You are also billed for any storage transactions involved in the restore process.

For more information about pricing for point-in-time restore, see [Block blob pricing](#).

## Next steps

- [Perform a point-in-time restore on block blob data](#)
- [Change feed support in Azure Blob Storage](#)
- [Enable soft delete for blobs](#)
- [Enable and manage blob versioning](#)

# Blob snapshots

8/22/2022 • 9 minutes to read • [Edit Online](#)

A snapshot is a read-only version of a blob that's taken at a point in time.

## NOTE

Blob versioning offers a superior way to maintain previous versions of a blob. For more information, see [Blob versioning](#).

## About blob snapshots

### IMPORTANT

Snapshots in accounts that have the hierarchical namespace feature enabled is currently in PREVIEW. See the [Supplemental Terms of Use for Microsoft Azure Previews](#) for legal terms that apply to Azure features that are in beta, preview, or otherwise not yet released into general availability.

To enroll in the preview, see [this form](#).

A snapshot of a blob is identical to its base blob, except that the blob URI has a **DateTime** value appended to the blob URI to indicate the time at which the snapshot was taken. For example, if a page blob URL is

`http://storagesample.core.blob.windows.net/mydrives/myvhds`, the snapshot URL is similar to

`http://storagesample.core.blob.windows.net/mydrives/myvhds?snapshot=2011-03-09T01:42:34.9360000Z`.

## NOTE

All snapshots share the base blob's URI. The only distinction between the base blob and the snapshot is the appended **DateTime** value.

A blob can have any number of snapshots. Snapshots persist until they are explicitly deleted, either independently or as part of a [Delete Blob](#) operation for the base blob. You can enumerate the snapshots associated with the base blob to track your current snapshots.

When you create a snapshot of a blob, the blob's system properties are copied to the snapshot with the same values. The base blob's metadata is also copied to the snapshot, unless you specify separate metadata for the snapshot when you create it. After you create a snapshot, you can read, copy, or delete it, but you cannot modify it.

Any leases associated with the base blob do not affect the snapshot. You cannot acquire a lease on a snapshot.

You can create a snapshot of a blob in the Hot or Cool tier. Snapshots on blobs in the Archive tier are not supported.

A VHD file is used to store the current information and status for a VM disk. You can detach a disk from within the VM or shut down the VM, and then take a snapshot of its VHD file. You can use that snapshot file later to retrieve the VHD file at that point in time and recreate the VM.

## Pricing and billing

Creating a snapshot, which is a read-only copy of a blob, can result in additional data storage charges to your

account. When designing your application, it is important to be aware of how these charges might accrue so that you can minimize costs.

Blob snapshots, like blob versions, are billed at the same rate as active data. How snapshots are billed depends on whether you have explicitly set the tier for the base blob or for any of its snapshots (or versions). For more information about blob tiers, see [Hot, Cool, and Archive access tiers for blob data](#).

If you have not changed a blob or snapshot's tier, then you are billed for unique blocks of data across that blob, its snapshots, and any versions it may have. For more information, see [Billing when the blob tier has not been explicitly set](#).

If you have changed a blob or snapshot's tier, then you are billed for the entire object, regardless of whether the blob and snapshot are eventually in the same tier again. For more information, see [Billing when the blob tier has been explicitly set](#).

For more information about billing details for blob versions, see [Blob versioning](#).

### Minimize costs with snapshot management

Microsoft recommends managing your snapshots carefully to avoid extra charges. You can follow these best practices to help minimize the costs incurred by the storage of your snapshots:

- Delete and re-create snapshots associated with a blob whenever you update the blob, even if you are updating with identical data, unless your application design requires that you maintain snapshots. By deleting and re-creating the blob's snapshots, you can ensure that the blob and snapshots do not diverge.
- If you are maintaining snapshots for a blob, avoid calling methods that overwrite the entire blob when you update the blob. Instead, update the fewest possible number of blocks in order to keep costs low.

### Billing when the blob tier has not been explicitly set

If you have not explicitly set the blob tier for a base blob or any of its snapshots, then you are charged for unique blocks or pages across the blob, its snapshots, and any versions it may have. Data that is shared across a blob and its snapshots is charged only once. When a blob is updated, then data in a base blob diverges from the data stored in its snapshots, and the unique data is charged per block or page.

When you replace a block within a block blob, that block is subsequently charged as a unique block. This is true even if the block has the same block ID and the same data as it has in the snapshot. After the block is committed again, it diverges from its counterpart in the snapshot, and you will be charged for its data. The same holds true for a page in a page blob that's updated with identical data.

Blob storage does not have a means to determine whether two blocks contain identical data. Each block that is uploaded and committed is treated as unique, even if it has the same data and the same block ID. Because charges accrue for unique blocks, it's important to keep in mind that updating a blob when that blob has snapshots or versions will result in additional unique blocks and additional charges.

When a blob has snapshots, call update operations on block blobs so that they update the least possible number of blocks. The write operations that permit fine-grained control over blocks are [Put Block](#) and [Put Block List](#). The [Put Blob](#) operation, on the other hand, replaces the entire contents of a blob and so may lead to additional charges.

The following scenarios demonstrate how charges accrue for a block blob and its snapshots when the blob tier has not been explicitly set.

#### Scenario 1

In scenario 1, the base blob has not been updated after the snapshot was taken, so charges are incurred only for unique blocks 1, 2, and 3.

	Base Blob	Snapshot
ID = 1	AAA	ID = 1
ID = 2	BBB	ID = 2
ID = 3	CCC	ID = 3

#### Scenario 2

In scenario 2, the base blob has been updated, but the snapshot has not. Block 3 was updated, and even though it contains the same data and the same ID, it is not the same as block 3 in the snapshot. As a result, the account is charged for four blocks.

	Base Blob	Snapshot
ID = 1	AAA	ID = 1
ID = 2	BBB	ID = 2
ID = 3	CCC	ID = 3

#### Scenario 3

In scenario 3, the base blob has been updated, but the snapshot has not. Block 3 was replaced with block 4 in the base blob, but the snapshot still reflects block 3. As a result, the account is charged for four blocks.

	Base Blob	Snapshot
ID = 1	AAA	ID = 1
ID = 2	BBB	ID = 2
ID = 4	DDD	ID = 3

#### Scenario 4

In scenario 4, the base blob has been completely updated and contains none of its original blocks. As a result, the account is charged for all eight unique blocks.

	Base Blob	Snapshot 1	Snapshot 2
ID = 5	EEE	ID = 1	ID = 1
ID = 6	FFF	ID = 2	ID = 2
ID = 7	GGG	ID = 3	ID = 3
ID = 8	HHH		ID = 4

#### TIP

Avoid calling methods that overwrite the entire blob, and instead update individual blocks to keep costs low.

#### Billing when the blob tier has been explicitly set

If you have explicitly set the blob tier for a blob or snapshot (or version), then you are charged for the full content length of the object in the new tier, regardless of whether it shares blocks with an object in the original tier. You are also charged for the full content length of the oldest version in the original tier. Any versions or snapshots that remain in the original tier are charged for unique blocks that they may share, as described in [Billing when the blob tier has not been explicitly set](#).

#### Moving a blob to a new tier

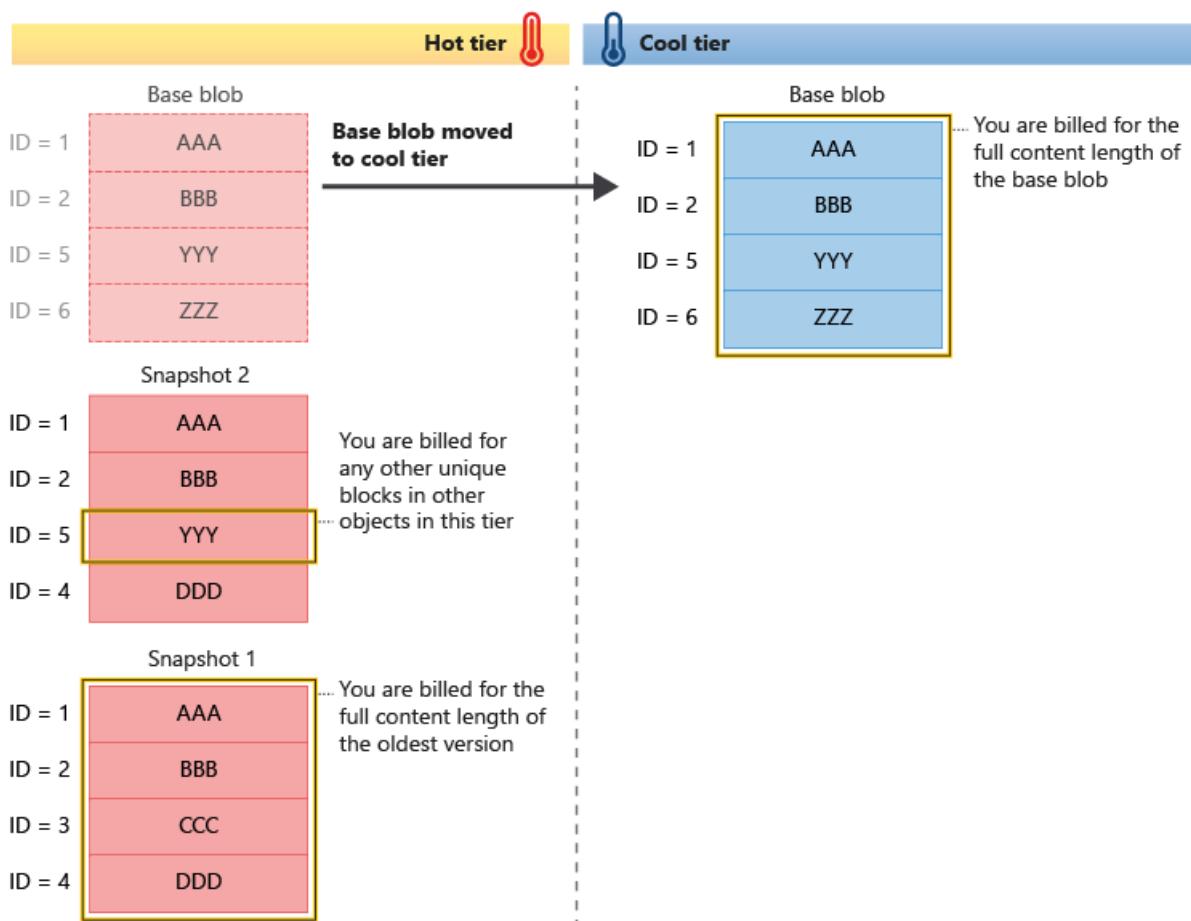
The following table describes the billing behavior for a blob or snapshot when it is moved to a new tier.

WHEN BLOB TIER IS SET EXPLICITLY ON...	THEN YOU ARE BILLED FOR...
A base blob with a snapshot	The base blob in the new tier and the oldest snapshot in the original tier, plus any unique blocks in other snapshots. <sup>1</sup>

WHEN BLOB TIER IS SET EXPLICITLY ON...	THEN YOU ARE BILLED FOR...
A base blob with a previous version and a snapshot	The base blob in the new tier, the oldest version in the original tier, and the oldest snapshot in the original tier, plus any unique blocks in other versions or snapshots <sup>1</sup> .
A snapshot	The snapshot in the new tier and the base blob in the original tier, plus any unique blocks in other snapshots. <sup>1</sup>

<sup>1</sup>If there are other previous versions or snapshots that have not been moved from their original tier, those versions or snapshots are charged based on the number of unique blocks they contain, as described in [Billing when the blob tier has not been explicitly set](#).

The following diagram illustrates how objects are billed when a blob with snapshots is moved to a different tier.



Explicitly setting the tier for a blob, version, or snapshot cannot be undone. If you move a blob to a new tier and then move it back to its original tier, you are charged for the full content length of the object even if it shares blocks with other objects in the original tier.

Operations that explicitly set the tier of a blob, version, or snapshot include:

- [Set Blob Tier](#)
- [Put Blob](#) with tier specified
- [Put Block List](#) with tier specified
- [Copy Blob](#) with tier specified

#### Deleting a blob when soft delete is enabled

When blob soft delete is enabled, if you delete or overwrite a base blob that has had its tier explicitly set, then any previous versions or snapshots of the soft-deleted blob are billed at full content length. For more information about how blob versioning and soft delete work together, see [Blob versioning and soft delete](#).

The following table describes the billing behavior for a blob that is soft-deleted, depending on whether versioning is enabled or disabled. When versioning is enabled, a new version is created when a blob is soft-deleted. When versioning is disabled, soft-deleting a blob creates a soft-delete snapshot.

WHEN YOU OVERWRITE A BASE BLOB WITH ITS TIER EXPLICITLY SET...	THEN YOU ARE BILLED FOR...
If blob soft delete and versioning are both enabled	All existing versions at full content length regardless of tier.
If blob soft delete is enabled but versioning is disabled	All existing soft-delete snapshots at full content length regardless of tier.

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## Next steps

- [Blob versioning](#)
- [Create and manage a blob snapshot in .NET](#)
- [Back up Azure unmanaged VM disks with incremental snapshots](#)

# Change feed support in Azure Blob Storage

8/22/2022 • 14 minutes to read • [Edit Online](#)

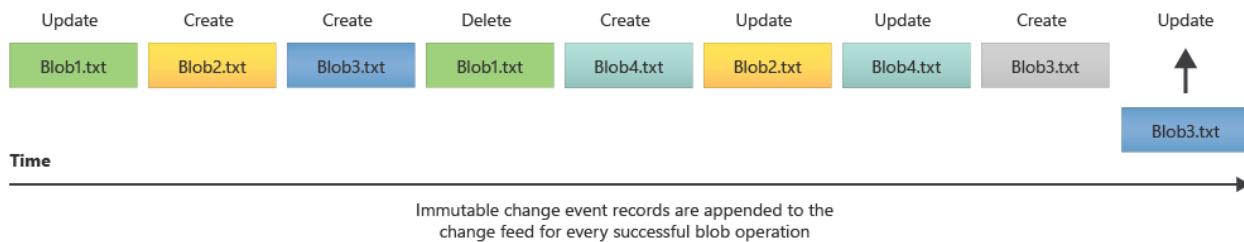
The purpose of the change feed is to provide transaction logs of all the changes that occur to the blobs and the blob metadata in your storage account. The change feed provides **ordered, guaranteed, durable, immutable, read-only** log of these changes. Client applications can read these logs at any time, either in streaming or in batch mode. 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 feed records are stored as [blobs](#) in a special container in your storage account at standard [blob pricing](#) cost. You can control the retention period of these files based on your requirements (See the [conditions](#) of the current release). 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.

You can process these logs asynchronously, incrementally or in-full. Any number of client applications can independently read the change feed, in parallel, and at their own pace. Analytics applications such as [Apache Drill](#) or [Apache Spark](#) can consume logs directly as Avro files, which let you process them at a low-cost, with high-bandwidth, and without having to write a custom application.

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.

Change feed is a prerequisite feature for [Object Replication](#) and [Point-in-time restore for block blobs](#).

#### NOTE

Change feed provides a durable, ordered log model of the changes that occur to a blob. Changes are written and made available in your change feed log within an order of a few minutes of the change. If your application has to react to events much quicker than this, consider using [Blob Storage events](#) instead. [Blob Storage Events](#) provides real-time one-time events which enable your Azure Functions or applications to quickly react to changes that occur to a blob.

## 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. You can enable and disable changes by using Azure Resource Manager templates on Portal or PowerShell.

Here's a few things to keep in mind when you enable the change feed.

- There's only one change feed for the blob service in each storage account. Change feed records are stored in the `$blobchangefeed` container.
- Create, Update, and Delete changes are captured only at the blob service level.
- The change feed captures *all* of the changes for all of the available events that occur on the account. Client applications can filter out event types as required. (See the [conditions](#) of the current release).
- Only standard general-purpose v2, premium block blob, and Blob storage accounts can enable the change feed. Accounts with a hierarchical namespace enabled are not currently supported. General-purpose v1 storage accounts are not supported but can be upgraded to general-purpose v2 with no downtime, see [Upgrade to a GPv2 storage account](#) for more information.
- [Portal](#)
- [PowerShell](#)
- [Template](#)

Enable change feed on your storage account by using Azure portal:

1. In the [Azure portal](#), select your storage account.
2. Navigate to the **Data protection** option under **Data Management**.
3. Under **Tracking**, select **Enable blob change feed**.
4. Choose the **Save** button to confirm your data protection settings.

The screenshot shows the Microsoft Azure Storage account settings for the 'contoso' storage account. The left sidebar lists various management options like Geo-replication, Data protection, Object replication, etc. The main pane shows the 'Data protection' section with several options under 'Recovery' and 'Tracking'. The 'Enable blob change feed' option under 'Tracking' is highlighted with a red box. A tooltip below it explains that it keeps track of create, modification, and delete changes to blobs.

## 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.

### NOTE

In the current release, the `$blobchangefeed` container is visible only in Azure portal but not visible in Azure Storage Explorer. You currently cannot see the `$blobchangefeed` container when you call ListContainers API but you are able to call the ListBlobs API directly on the container to see the blobs

Your client applications can consume the change feed by using the blob change feed processor library that is provided with the change feed processor SDK.

See [Process change feed logs in Azure Blob Storage](#).

## 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. To learn more, see the [Specifications](#).

An available hourly segment of the change feed is described in a manifest file that specifies the paths to the change feed files for that segment. The listing of the `$blobchangefeed/idx/segments/` virtual directory shows these segments ordered by time. The path of the segment describes the start of the hourly time-range that the segment represents. You can use that list to filter out the segments of logs that are of interest to you.

Name	Blob Type	Blob Tier	Length
Content Type			
-----	-----	-----	-----
\$blobchangefeed/idx/segments/1601/01/01/0000/meta.json	BlockBlob		584
application/json			
\$blobchangefeed/idx/segments/2019/02/22/1810/meta.json	BlockBlob		584
application/json			
\$blobchangefeed/idx/segments/2019/02/22/1910/meta.json	BlockBlob		584
application/json			
\$blobchangefeed/idx/segments/2019/02/23/0110/meta.json	BlockBlob		584
application/json			

#### NOTE

The `$blobchangefeed/idx/segments/1601/01/01/0000/meta.json` is automatically created when you enable the change feed. You can safely ignore this file. It is an always empty initialization file.

The segment manifest file (`meta.json`) shows the path of the change feed files for that segment in the `chunkFilePaths` property. Here's an example of a segment manifest file.

```
{
 "version": 0,
 "begin": "2019-02-22T18:10:00.000Z",
 "intervalSecs": 3600,
 "status": "Finalized",
 "config": {
 "version": 0,
 "configVersionEtag": "0x8d698f0fba563db",
 "numShards": 2,
 "recordsFormat": "avro",
 "formatSchemaVersion": 1,
 "shardDistFnVersion": 1
 },
 "chunkFilePaths": [
 "$blobchangefeed/log/00/2019/02/22/1810/",
 "$blobchangefeed/log/01/2019/02/22/1810/"
],
 "storageDiagnostics": {
 "version": 0,
 "lastModifiedTime": "2019-02-22T18:11:01.187Z",
 "data": {
 "aid": "55e507bf-8006-0000-00d9-ca346706b70c"
 }
 }
}
```

#### NOTE

The `$blobchangefeed` container appears only after you've enabled the change feed feature on your account. You'll have to wait a few minutes after you enable the change feed before you can list the blobs in the container.

## 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. The records are serialized and written to the file using the [Apache Avro](#) format specification. The records can be read by using the Avro file format specification. There are several libraries

available to process files in that format.

Change feed files are stored in the `$blobchangefeed/log/` virtual directory as [append blobs](#). The first change feed file under each path will have `00000` in the file name (For example `00000.avro`). The name of each subsequent log file added to that path will increment by 1 (For example: `00001.avro`).

## Event record schemas

For a description of each property, see [Azure Event Grid event schema for Blob Storage](#). The `BlobPropertiesUpdated` and `BlobSnapshotCreated` events are currently exclusive to change feed and not yet supported for Blob Storage Events.

### NOTE

The change feed files for a segment don't immediately appear after a segment is created. The length of delay is within the normal interval of publishing latency of the change feed which is within a few minutes of the change.

### Schema version 1

The following event types may be captured in the change feed records with schema version 1:

- `BlobCreated`
- `BlobDeleted`
- `BlobPropertiesUpdated`
- `BlobSnapshotCreated`

The following example shows a change event record in JSON format that uses event schema version 1:

```
{
 "schemaVersion": 1,
 "topic": "/subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>",
 "subject": "/blobServices/default/containers/<container>/blobs/<blob>",
 "eventType": "BlobCreated",
 "eventTime": "2022-02-17T12:59:41.400310Z",
 "id": "322343e3-8020-0000-00fe-233467066726",
 "data": {
 "api": "PutBlob",
 "clientRequestId": "f0270546-168e-4398-8fa8-107a1ac214d2",
 "requestId": "322343e3-8020-0000-00fe-233467000000",
 "etag": "0x8D9F2155CBF7928",
 "contentType": "application/octet-stream",
 "contentLength": 128,
 "blobType": "BlockBlob",
 "url": "https://www.myurl.com",
 "sequencer": "0000000000000001000000000000000020000000000000001d",
 "storageDiagnostics": {
 "bid": "9d725a00-8006-0000-00fe-233467000000",
 "seq": "(2,18446744073709551615,29,29)",
 "sid": "4cc94e71-f6be-75bf-e7b2-f9ac41458e5a"
 }
 }
}
```

### Schema version 3

The following event types may be captured in the change feed records with schema version 3:

- `BlobCreated`
- `BlobDeleted`
- `BlobPropertiesUpdated`
- `BlobSnapshotCreated`

The following example shows a change event record in JSON format that uses event schema version 3:

```
{
 "schemaVersion": 3,
 "topic": "/subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>",
 "subject": "/blobServices/default/containers/<container>/blobs/<blob>",
 "eventType": "BlobCreated",
 "eventTime": "2022-02-17T13:05:19.679824Z",
 "id": "eefe8fc8-8020-0000-00fe-23346706daaa",
 "data": {
 "api": "PutBlob",
 "clientRequestId": "00c0b6b7-bb67-4748-a3dc-86464863d267",
 "requestId": "eefe8fc8-8020-0000-00fe-233467000000",
 "etag": "0x8D9F216266170DC",
 "contentType": "application/octet-stream",
 "contentLength": 128,
 "blobType": "BlockBlob",
 "url": "https://www.myurl.com",
 "sequencer": "000000000000000100000000000000020000000000000001",
 "previousInfo": {
 "SoftDeleteSnapshot": "2022-02-17T13:08:42.4825913Z",
 "WasBlobSoftDeleted": "true",
 "BlobVersion": "2024-02-17T16:11:52.0781797Z",
 "LastVersion": "2022-02-17T16:11:52.0781797Z",
 "PreviousTier": "Hot"
 },
 "snapshot": "2022-02-17T16:09:16.7261278Z",
 "blobPropertiesUpdated": {
 "ContentLanguage": {
 "current": "pl-PL",
 "previous": "nl-NL"
 },
 "CacheControl": {
 "current": "max-age=100",
 "previous": "max-age=99"
 },
 "ContentEncoding": {
 "current": "gzip, identity",
 "previous": "gzip"
 },
 "ContentMD5": {
 "current": "Q2h1Y2sgSW51ZwDIAXR5IQ==",
 "previous": "Q2h1Y2sgSW=="
 },
 "ContentDisposition": {
 "current": "attachment",
 "previous": ""
 },
 "ContentType": {
 "current": "application/json",
 "previous": "application/octet-stream"
 }
 },
 "storageDiagnostics": {
 "bid": "9d726370-8006-0000-00ff-233467000000",
 "seq": "(2,18446744073709551615,29,29)",
 "sid": "4cc94e71-f6be-75bf-e7b2-f9ac41458e5a"
 }
 }
}
```

Schema version 4

The following event types may be captured in the change feed records with schema version 4:

- BlobCreated

- BlobDeleted
  - BlobPropertiesUpdated
  - BlobSnapshotCreated
  - BlobTierChanged
  - BlobAsyncOperationInitiated
  - RestorePointMarkerCreated

The following example shows a change event record in JSON format that uses event schema version 4:

```
{
 "schemaVersion": 4,
 "topic": "/subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>",
 "subject": "/blobServices/default/containers/<container>/blobs/<blob>",
 "eventType": "BlobCreated",
 "eventTime": "2022-02-17T13:08:42.483590Z",
 "id": "ca76bce1-8020-0000-0fff-23346706e769",
 "data": {
 "api": "PutBlob",
 "clientRequestId": "58fbfee9-6cf5-4096-9666-c42980beee65",
 "requestId": "ca76bce1-8020-0000-0fff-233467000000",
 "etag": "0x8D9F2169F42D701",
 "contentType": "application/octet-stream",
 "contentLength": 128,
 "blobType": "BlockBlob",
 "blobVersion": "2022-02-17T16:11:52.5901564Z",
 "containerVersion": "0000000000000001",
 "blobTier": "Archive",
 "url": "https://www.myurl.com",
 "sequencer": "000000000000000100000000000000020000000000000001d",
 "previousInfo": {
 "SoftDeleteSnapshot": "2022-02-17T13:08:42.4825913Z",
 "WasBlobSoftDeleted": "true",
 "BlobVersion": "2024-02-17T16:11:52.0781797Z",
 "LastVersion" : "2022-02-17T16:11:52.0781797Z",
 "PreviousTier": "Hot"
 },
 "snapshot": "2022-02-17T16:09:16.7261278Z",
 "blobPropertiesUpdated" : {
 "ContentLanguage" : {
 "current" : "pl-PL",
 "previous" : "nl-NL"
 },
 "CacheControl" : {
 "current" : "max-age=100",
 "previous" : "max-age=99"
 },
 "ContentEncoding" : {
 "current" : "gzip, identity",
 "previous" : "gzip"
 },
 "ContentMD5" : {
 "current" : "Q2h1Y2sgSW51ZwDIAXR5IQ==",
 "previous" : "Q2h1Y2sgSW="
 },
 "ContentDisposition" : {
 "current" : "attachment",
 "previous" : ""
 },
 "ContentType" : {
 "current" : "application/json",
 "previous" : "application/octet-stream"
 }
 },
 "asyncOperationInfo": {
 }
```

```
 "DestinationTier": "Hot",
 "WasAsyncOperation": "true",
 "CopyId": "copyId"
 },
 "storageDiagnostics": {
 "bid": "9d72687f-8006-0000-00ff-233467000000",
 "seq": "(2,18446744073709551615,29,29)",
 "sid": "4cc94e71-f6be-75bf-e7b2-f9ac41458e5a"
 }
}
```

## Schema version 5

The following event types may be captured in the change feed records with schema version 5:

- BlobCreated
  - BlobDeleted
  - BlobPropertiesUpdated
  - BlobSnapshotCreated
  - BlobTierChanged
  - BlobAsyncOperationInitiated

The following example shows a change event record in JSON format that uses event schema version 5:

```
{
 "schemaVersion": 5,
 "topic": "/subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>",
 "subject": "/blobServices/default/containers/<container>/blobs/<blob>",
 "eventType": "BlobCreated",
 "eventTime": "2022-02-17T13:12:11.5746587Z",
 "id": "62616073-8020-0000-00ff-233467060cc0",
 "data": {
 "api": "PutBlob",
 "clientRequestId": "b3f9b39a-ae5a-45ac-afad-95ac9e9f2791",
 "requestId": "62616073-8020-0000-00ff-233467000000",
 "etag": "0x8D9F2171BE32588",
 "contentType": "application/octet-stream",
 "contentLength": 128,
 "blobType": "BlockBlob",
 "blobVersion": "2022-02-17T16:11:52.5901564Z",
 "containerVersion": "0000000000000001",
 "blobTier": "Archive",
 "url": "https://www.myurl.com",
 "sequencer": "0000000000000001000000000000000020000000000000001d",
 "previousInfo": {
 "SoftDeleteSnapshot": "2022-02-17T13:12:11.5726507Z",
 "WasBlobSoftDeleted": "true",
 "BlobVersion": "2024-02-17T16:11:52.0781797Z",
 "LastVersion" : "2022-02-17T16:11:52.0781797Z",
 "PreviousTier": "Hot"
 },
 "snapshot" : "2022-02-17T16:09:16.7261278Z",
 "blobPropertiesUpdated" : {
 "ContentLanguage" : {
 "current" : "pl-PL",
 "previous" : "nl-NL"
 },
 "CacheControl" : {
 "current" : "max-age=100",
 "previous" : "max-age=99"
 },
 "ContentEncoding" : {
 "current" : "gzip, identity",
 "previous" : "gzip"
 }
 }
 }
}
```

```

 previous : blob
},
"ContentMD5" : {
 "current" : "Q2h1Y2sgSW51ZwDIAXR5IQ==",
 "previous" : "Q2h1Y2sgSW="
},
"ContentDisposition" : {
 "current" : "attachment",
 "previous" : ""
},
"ContentType" : {
 "current" : "application/json",
 "previous" : "application/octet-stream"
}
},
"asyncOperationInfo": {
 "DestinationTier": "Hot",
 "WasAsyncOperation": "true",
 "CopyId": "copyId"
},
"blobTagsUpdated": {
 "previous": {
 "Tag1": "Value1_3",
 "Tag2": "Value2_3"
 },
 "current": {
 "Tag1": "Value1_4",
 "Tag2": "Value2_4"
 }
},
"restorePointMarker": {
 "rpi": "cbd73e3d-f650-4700-b90c-2f067bce639c",
 "rpp": "cbd73e3d-f650-4700-b90c-2f067bce639c",
 "rpl": "test-restore-label",
 "rpt": "2022-02-17T13:56:09.3559772Z"
},
"storageDiagnostics": {
 "bid": "9d726db1-8006-0000-00ff-233467000000",
 "seq": "(2,18446744073709551615,29,29)",
 "sid": "4cc94e71-f6be-75bf-e7b2-f9ac41458e5a"
}
}
}
}

```

## Specifications

- Change events records are only appended to the change feed. Once these records are appended, they are immutable and record-position is stable. Client applications can maintain their own checkpoint on the read position of the change feed.
- Change event records are appended within an order of few minutes of the change. Client applications can choose to consume records as they are appended for streaming access or in bulk at any other time.
- Change event records are ordered by modification order **per blob**. Order of changes across blobs is undefined in Azure Blob Storage. All changes in a prior segment are before any changes in subsequent segments.
- Change event records are serialized into the log file by using the [Apache Avro 1.8.2](#) format specification.
- Change event records where the `eventType` has a value of `Control` are internal system records and don't reflect a change to objects in your account. You can safely ignore those records.
- Values in the `storageDiagnostics` property bag are for internal use only and not designed for use by your application. Your applications shouldn't have a contractual dependency on that data. You can safely ignore

those properties.

- The time represented by the segment is **approximate** with bounds of 15 minutes. So to ensure consumption of all records within a specified time, consume the consecutive previous and next hour segment.
- Each segment can have a different number of `chunkFilePaths` due to internal partitioning of the log stream to manage publishing throughput. The log files in each `chunkFilePath` are guaranteed to contain mutually exclusive blobs, and can be consumed and processed in parallel without violating the ordering of modifications per blob during the iteration.
- The Segments start out in `Publishing` status. Once the appending of the records to the segment is complete, it will be `Finalized`. Log files in any segment that is dated after the date of the `LastConsumable` property in the `$blobchangefeed/meta/Segments.json` file, should not be consumed by your application. Here's an example of the `LastConsumable` property in a `$blobchangefeed/meta/Segments.json` file:

```
{
 "version": 0,
 "lastConsumable": "2019-02-23T01:10:00.000Z",
 "storageDiagnostics": {
 "version": 0,
 "lastModifiedTime": "2019-02-23T02:24:00.556Z",
 "data": {
 "aid": "55e551e3-8006-0000-00da-ca346706bfe4",
 "lfz": "2019-02-22T19:10:00.000Z"
 }
 }
}
```

## Conditions and known issues

This section describes known issues and conditions in the current release of the change feed.

- The `url` property of the log file is currently always empty.
- The `LastConsumable` property of the segments.json file does not list the very first segment that the change feed finalizes. This issue occurs only after the first segment is finalized. All subsequent segments after the first hour are accurately captured in the `LastConsumable` property.
- You currently cannot see the `$blobchangefeed` container when you call ListContainers API and the container does not show up on Azure portal or Storage Explorer. You can view the contents by calling the ListBlobs API on the `$blobchangefeed` container directly.
- Storage accounts that have previously initiated an [account failover](#) may have issues with the log file not appearing. Any future account failovers may also impact the log file.

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## FAQ

**What is the difference between the change feed and Storage Analytics logging?**

Analytics logs have records of all read, write, list, and delete operations with successful and failed requests across all operations. Analytics logs are best-effort and no ordering is guaranteed.

The change feed is a solution that provides transactional log of successful mutations or changes to your account such as blob creation, modification, and deletions. The change feed guarantees all events to be recorded and displayed in the order of successful changes per blob, thus you do not have to filter out noise from a huge volume of read operations or failed requests. The change feed is fundamentally designed and optimized for application development that require certain guarantees.

### Should I use the change feed or Storage events?

You can leverage both features as the change feed and [Blob storage events](#) provide the same information with the same delivery reliability guarantee, with the main difference being the latency, ordering, and storage of event records. The change feed publishes records to the log within few minutes of the change and also guarantees the order of change operations per blob. Storage events are pushed in real time and might not be ordered. Change feed events are durably stored inside your storage account as read-only stable logs with your own defined retention, while storage events are transient to be consumed by the event handler unless you explicitly store them. With change feed, any number of your applications can consume the logs at their own convenience using blob APIs or SDKs.

## Next steps

- See an example of how to read the change feed by using a .NET client application. See [Process change feed logs in Azure Blob Storage](#).
- Learn about how to react to events in real time. See [Reacting to Blob Storage events](#)
- Learn more about detailed logging information for both successful and failed operations for all requests. See [Azure Storage analytics logging](#)

# Store business-critical blob data with immutable storage

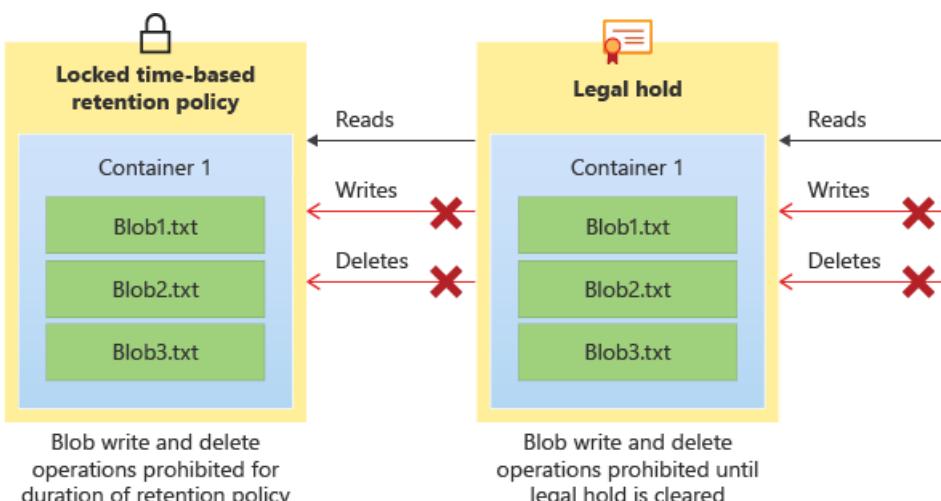
8/22/2022 • 11 minutes to read • [Edit Online](#)

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 cannot 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. To learn more about time-based retention policies, see [Time-based retention policies for immutable blob data](#).
- **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. To learn more about legal hold policies, see [Legal holds for immutable blob data](#).

The following diagram shows how time-based retention policies and legal holds prevent write and delete operations while they are in effect.



## About immutable storage for blobs

Immutable storage helps healthcare organizations, financial institutions, and related industries—particularly broker-dealer organizations—to store data securely. Immutable storage can be leveraged in any scenario to protect critical data against modification or deletion.

Typical applications include:

- **Regulatory compliance:** Immutable storage for Azure Blob Storage helps organizations address SEC 17a-4(f), CFTC 1.31(d), FINRA, and other regulations.
- **Secure document retention:** Immutable storage for blobs ensures that data can't be modified or deleted by any user, not even by users with account administrative privileges.
- **Legal hold:** Immutable storage for blobs enables users to store sensitive information that is critical to

litigation or business use in a tamper-proof state for the desired duration until the hold is removed. This feature is not limited only to legal use cases but can also be thought of as an event-based hold or an enterprise lock, where the need to protect data based on event triggers or corporate policy is required.

## Regulatory compliance

Microsoft retained a leading independent assessment firm that specializes in records management and information governance, Cohasset Associates, to evaluate immutable storage for blobs and its compliance with requirements specific to the financial services industry. Cohasset validated that immutable storage, when used to retain blobs in a WORM state, meets the relevant storage requirements of CFTC Rule 1.31(c)-(d), FINRA Rule 4511, and SEC Rule 17a-4(f). Microsoft targeted this set of rules because they represent the most prescriptive guidance globally for records retention for financial institutions.

The Cohasset report is available in the [Microsoft Service Trust Center](#). The [Azure Trust Center](#) contains detailed information about Microsoft's compliance certifications. To request a letter of attestation from Microsoft regarding WORM immutability compliance, please contact [Azure Support](#).

## Immutability policy scope

Immutability policies can be scoped to a blob version or to a container. How an object behaves under an immutability policy depends on the scope of the policy. For more information about policy scope for each type of immutability policy, see the following sections:

- [Time-based retention policy scope](#)
- [Legal hold scope](#)

You can configure both a time-based retention policy and a legal hold for a resource (container or blob version), depending on the scope.

### Version-level scope

To configure an immutability policy that is scoped to a blob version, you must enable support for version-level immutability on either the storage account or a container. After you enable support for version-level immutability on a storage account, you can configure a default policy at the account level that applies to all objects subsequently created in the storage account. If you enable support for version-level immutability on an individual container, you can configure a default policy for that container that applies to all objects subsequently created in the container.

The following table summarizes which immutability policies are supported for each resource scope:

RESOURCE	ENABLE VERSION-LEVEL IMMUTABILITY POLICIES	POLICY SUPPORT
Account	Yes, at account creation only.	Supports one default version-level immutability policy. The default policy applies to any new blob versions created in the account after the policy is configured.  Does not support legal hold.
Container	Yes, at container creation. Existing containers must be migrated to support version-level immutability policies.	Supports one default version-level immutability policy. The default policy applies to any new blob versions created in the container after the policy is configured.  Does not support legal hold.

RESOURCE	ENABLE VERSION-LEVEL IMMUTABILITY POLICIES	POLICY SUPPORT
----------	--------------------------------------------	----------------

Blob version	N/A	Supports one version-level immutability policy and one legal hold. A policy on a blob version can override a default policy specified on the account or container.
--------------	-----	--------------------------------------------------------------------------------------------------------------------------------------------------------------------

### Container-level scope

When support for version-level immutability policies has not been enabled for a storage account or a container, then any immutability policies are scoped to the container. A container supports one immutability policy and one legal hold. Policies apply to all objects within the container.

## Summary of immutability scenarios

The protection afforded by an immutability policy depends on the scope of the immutability policy and, in the case of a time-based retention policy, whether it is locked or unlocked and whether it is active or expired.

### Scenarios with version-level scope

The following table provides a summary of protections provided by version-level immutability policies.

SCENARIO	PROHIBITED OPERATIONS	BLOB PROTECTION	CONTAINER PROTECTION	ACCOUNT PROTECTION
A blob version is protected by an <i>active</i> retention policy and/or a legal hold is in effect	<a href="#">Delete Blob</a> , <a href="#">Set Blob Metadata</a> , <a href="#">Put Page</a> , and <a href="#">Append Block</a> <sup>1</sup>	The blob version cannot be deleted. User metadata cannot be written.  Overwriting a blob with <a href="#">Put Blob</a> , <a href="#">Put Block List</a> , or <a href="#">Copy Blob</a> creates a new version. <sup>2</sup>	Container deletion fails if at least one blob exists in the container, regardless of whether policy is locked or unlocked.	Storage account deletion fails if there is at least one container with version-level immutable storage enabled, or if it is enabled for the account.
A blob version is protected by an <i>expired</i> retention policy and no legal hold is in effect	<a href="#">Set Blob Metadata</a> , <a href="#">Put Page</a> , and <a href="#">Append Block</a> <sup>1</sup>	The blob version can be deleted. User metadata cannot be written.  Overwriting a blob with <a href="#">Put Blob</a> , <a href="#">Put Block List</a> , or <a href="#">Copy Blob</a> creates a new version <sup>2</sup> .	Container deletion fails if at least one blob exists in the container, regardless of whether policy is locked or unlocked.	Storage account deletion fails if there is at least one container that contains a blob version with a locked time-based retention policy.  Unlocked policies do not provide delete protection.

<sup>1</sup> The [Append Block](#) operation is only permitted for time-based retention policies with the `allowProtectedAppendWrites` property enabled. For more information, see [Allow protected append blobs](#)

writes.<sup>2</sup> Blob versions are always immutable for content. If versioning is enabled for the storage account, then a write operation to a block blob creates a new version, with the exception of the [Put Block](#) operation.

## Scenarios with container-level scope

The following table provides a summary of protections provided by container-level immutability policies.

SCENARIO	PROHIBITED OPERATIONS	BLOB PROTECTION	CONTAINER PROTECTION	ACCOUNT PROTECTION
A container is protected by an <i>active</i> time-based retention policy with container scope and/or a legal hold is in effect	<a href="#">Delete Blob</a> , <a href="#">Put Blob<sup>1</sup></a> , <a href="#">Set Blob Metadata</a> , <a href="#">Put Page</a> , <a href="#">Set Blob Properties</a> , <a href="#">Snapshot Blob</a> , <a href="#">Incremental Copy Blob</a> , <a href="#">Append Block<sup>2</sup></a>	All blobs in the container are immutable for content and user metadata	Container deletion fails if a container-level policy is in effect.	Storage account deletion fails if there is a container with at least one blob present.
A container is protected by an <i>expired</i> time-based retention policy with container scope and no legal hold is in effect	<a href="#">Put Blob<sup>1</sup></a> , <a href="#">Set Blob Metadata</a> , <a href="#">Put Page</a> , <a href="#">Set Blob Properties</a> , <a href="#">Snapshot Blob</a> , <a href="#">Incremental Copy Blob</a> , <a href="#">Append Block<sup>2</sup></a>	Delete operations are allowed. Overwrite operations are not allowed.	Container deletion fails if at least one blob exists in the container, regardless of whether policy is locked or unlocked.	Storage account deletion fails if there is at least one container with a locked time-based retention policy.  Unlocked policies do not provide delete protection.

<sup>1</sup> Azure Storage permits the [Put Blob](#) operation to create a new blob. Subsequent overwrite operations on an existing blob path in an immutable container are not allowed.

<sup>2</sup> The [Append Block](#) operation is only permitted for time-based retention policies with the `allowProtectedAppendWrites` property enabled. For more information, see [Allow protected append blobs writes](#).

### NOTE

Some workloads, such as [SQL Backup to URL](#), create a blob and then add to it. If a container has an active time-based retention policy or legal hold in place, this pattern will not succeed.

## Supported account configurations

Immutability policies are supported for both new and existing storage accounts. The following table shows which types of storage accounts are supported for each type of policy:

TYPE OF IMMUTABILITY POLICY	SCOPE OF POLICY	TYPES OF STORAGE ACCOUNTS SUPPORTED	SUPPORTS HIERARCHICAL NAMESPACE (PREVIEW)
Time-based retention policy	Version-level scope	General-purpose v2 Premium block blob	No
Time-based retention policy	Container-level scope	General-purpose v2 Premium block blob General-purpose v1 (legacy) <sup>1</sup> Blob storage (legacy)	Yes

TYPE OF IMMUTABILITY POLICY	SCOPE OF POLICY	TYPES OF STORAGE ACCOUNTS SUPPORTED	SUPPORTS HIERARCHICAL NAMESPACE (PREVIEW)
Legal hold	Version-level scope	General-purpose v2 Premium block blob	No
Legal hold	Container-level scope	General-purpose v2 Premium block blob General-purpose v1 (legacy) <sup>1</sup> Blob storage (legacy)	Yes

<sup>1</sup> Microsoft recommends upgrading general-purpose v1 accounts to general-purpose v2 so that you can take advantage of more features. For information on upgrading an existing general-purpose v1 storage account, see [Upgrade a storage account](#).

## Access tiers

All blob access tiers support immutable storage. You can change the access tier of a blob with the Set Blob Tier operation. For more information, see [Hot, Cool, and Archive access tiers for blob data](#).

## Redundancy configurations

All redundancy configurations support immutable storage. For geo-redundant configurations, customer-managed failover is not supported. For more information about redundancy configurations, see [Azure Storage redundancy](#).

## Hierarchical namespace support

Immutable storage support for accounts with a hierarchical namespace is in preview. To enroll in the preview, see [this form](#).

Keep in mind that you cannot rename or move a blob when the blob is in the immutable state and the account has a hierarchical namespace enabled. Both the blob name and the directory structure provide essential container-level data that cannot be modified once the immutable policy is in place.

### IMPORTANT

Immutable storage for Azure Blob Storage in accounts that have the hierarchical namespace feature enabled is currently in PREVIEW. See the [Supplemental Terms of Use for Microsoft Azure Previews](#) for legal terms that apply to Azure features that are in beta, preview, or otherwise not yet released into general availability.

## Recommended blob types

Microsoft recommends that you configure immutability policies mainly for block blobs and append blobs. Configuring an immutability policy for a page blob that stores a VHD disk for an active virtual machine is discouraged as writes to the disk will be blocked. Microsoft recommends that you thoroughly review the documentation and test your scenarios before locking any time-based policies.

## Immutable storage with blob soft delete

When blob soft delete is configured for a storage account, it applies to all blobs within the account regardless of whether a legal hold or time-based retention policy is in effect. Microsoft recommends enabling soft delete for additional protection before any immutability policies are applied.

If you enable blob soft delete and then configure an immutability policy, any blobs that have already been soft deleted will be permanently deleted once the soft delete retention policy has expired. Soft-deleted blobs can be restored during the soft delete retention period. A blob or version that has not yet been soft deleted is protected

by the immutability policy and cannot be soft deleted until after the time-based retention policy has expired or the legal hold has been removed.

## Use blob inventory to track immutability policies

Azure Storage blob inventory provides an overview of the containers in your storage accounts and the blobs, snapshots, and blob versions within them. You can use the blob inventory report to understand the attributes of blobs and containers, including whether a resource has an immutability policy configured.

When you enable blob inventory, Azure Storage generates an inventory report on a daily basis. The report provides an overview of your data for business and compliance requirements.

For more information about blob inventory, see [Azure Storage blob inventory \(preview\)](#).

## Pricing

There is no additional capacity charge for using immutable storage. Immutable data is priced in the same way as mutable data. For pricing details on Azure Blob Storage, see the [Azure Storage pricing page](#).

Creating, modifying, or deleting a time-based retention policy or legal hold on a blob version results in a write transaction charge.

If you fail to pay your bill and your account has an active time-based retention policy in effect, normal data retention policies will apply as stipulated in the terms and conditions of your contract with Microsoft. For general information, see [Data management at Microsoft](#).

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## Next steps

- [Data protection overview](#)
- [Time-based retention policies for immutable blob data](#)
- [Legal holds for immutable blob data](#)
- [Configure immutability policies for blob versions](#)
- [Configure immutability policies for containers](#)

# Time-based retention policies for immutable blob data

8/22/2022 • 11 minutes to read • [Edit Online](#)

A time-based retention policy stores blob data in a Write-Once, Read-Many (WORM) format for a specified interval. When a time-based retention policy is set, clients can create and read blobs, but cannot modify or delete them. After the retention interval has expired, blobs can be deleted but not overwritten.

For more information about immutability policies for Blob Storage, see [Store business-critical blob data with immutable storage](#).

## Retention interval for a time-based policy

The minimum retention interval for a time-based retention policy is one day, and the maximum is 146,000 days (400 years).

When you configure a time-based retention policy, the affected objects will stay in the immutable state for the duration of the *effective* retention period. The effective retention period for objects is equal to the difference between the blob's creation time and the user-specified retention interval. Because a policy's retention interval can be extended, immutable storage uses the most recent value of the user-specified retention interval to calculate the effective retention period.

For example, suppose that a user creates a time-based retention policy with a retention interval of five years. An existing blob in that container, *testblob1*, was created one year ago, so the effective retention period for *testblob1* is four years. When a new blob, *testblob2*, is uploaded to the container, the effective retention period for *testblob2* is five years from the time of its creation.

## Locked versus unlocked policies

When you first configure a time-based retention policy, the policy is unlocked for testing purposes. When you have finished testing, you can lock the policy so that it is fully compliant with SEC 17a-4(f) and other regulatory compliance.

Both locked and unlocked policies protect against deletes and overwrites. However, you can modify an unlocked policy by shortening or extending the retention period. You can also delete an unlocked policy.

You cannot delete a locked time-based retention policy. You can extend the retention period, but you cannot decrease it. A maximum of five increases to the effective retention period is allowed over the lifetime of a locked policy that is defined at the container level. For a policy configured for a blob version, there is no limit to the number of increase to the effective period.

### IMPORTANT

A time-based retention policy must be locked for the blob to be in a compliant immutable (write and delete protected) state for SEC 17a-4(f) and other regulatory compliance. Microsoft recommends that you lock the policy in a reasonable amount of time, typically less than 24 hours. While the unlocked state provides immutability protection, using the unlocked state for any purpose other than short-term testing is not recommended.

## Time-based retention policy scope

A time-based retention policy can be configured at either of the following scopes:

- Version-level policy: A time-based retention policy can be configured to apply to a blob version for granular management of sensitive data. You can apply the policy to an individual version, or configure a default policy for a storage account or individual container that will apply by default to all blobs uploaded to that account or container.
- Container-level policy: A time-based retention policy that is configured at the container level applies to all objects in that container. Individual objects cannot be configured with their own immutability policies.

Audit logs are available on the container for both version-level and container-level time-based retention policies. Audit logs are not available for a policy that is scoped to a blob version.

### **Version-level policy scope**

To configure version-level retention policies, you must first enable version-level immutability on the storage account or parent container. Version-level immutability cannot be disabled after it is enabled, although unlocked policies can be deleted. For more information, see [Enable support for version-level immutability](#).

Version-level immutability on the storage account must be enabled when you create the account. When you enable version-level immutability for a new storage account, all containers subsequently created in that account automatically support version-level immutability. It's not possible to disable support for version-level immutability on a storage account after you've enabled it, nor is it possible to create a container without version-level immutability support when it's enabled for the account.

If you have not enabled support for version-level immutability on the storage account, then you can enable support for version-level immutability on an individual container at the time that you create the container. Existing containers can also support version-level immutability, but must undergo a migration process first. This process may take some time and is not reversible. You can migrate ten containers at a time per storage account. For more information about migrating a container to support version-level immutability, see [Migrate an existing container to support version-level immutability](#).

Version-level time-based retention policies require that [blob versioning](#) is enabled for the storage account. To learn how to enable blob versioning, see [Enable and manage blob versioning](#). Keep in mind that enabling versioning may have a billing impact. For more information, see the [Pricing and billing](#) section in [Blob versioning](#).

After versioning is enabled, when a blob is first uploaded, that version of the blob is the current version. Each time the blob is overwritten, a new version is created that stores the previous state of the blob. When you delete the current version of a blob, the current version becomes a previous version and is retained until explicitly deleted. A previous blob version possesses the time-based retention policy that was in effect when the current version became a previous version.

If a default policy is in effect for the storage account or container, then when an overwrite operation creates a previous version, the new current version inherits the default policy for the account or container.

Each version may have only one time-based retention policy configured. A version may also have one legal hold configured. For more details about supported immutability policy configurations based on scope, see [Immutability policy scope](#).

To learn how to configure version-level time-based retention policies, see [Configure immutability policies for blob versions](#).

### **Configure a policy on the current version**

After you enable support for version-level immutability for a storage account or container, then you have the option to configure a default time-based retention policy for the account or container. When you configure a default time-based retention policy for the account or container and then upload a blob, the blob inherits that default policy. You can also choose to override the default policy for any blob on upload by configuring a custom

policy for that blob.

If the default time-based retention policy for the account or container is unlocked, then the current version of a blob that inherits the default policy will also have an unlocked policy. After an individual blob is uploaded, you can shorten or extend the retention period for the policy on the current version of the blob, or delete the current version. You can also lock the policy for the current version, even if the default policy on the account or container remains unlocked.

If the default time-based retention policy for the account or container is locked, then the current version of a blob that inherits the default policy will also have an locked policy. However, if you override the default policy when you upload a blob by setting a policy only for that blob, then that blob's policy will remain unlocked until you explicitly lock it. When the policy on the current version is locked, you can extend the retention interval, but you cannot delete the policy or shorten the retention interval.

If there is no default policy configured for either the storage account or the container, then you can upload a blob either with a custom policy or with no policy.

If the default policy on a storage account or container is modified, policies on objects within that container remain unchanged, even if those policies were inherited from the default policy.

The following table shows the various options available for setting a time-based retention policy on a blob on upload:

DEFAULT POLICY STATUS ON ACCOUNT OR CONTAINER	UPLOAD A BLOB WITH THE DEFAULT POLICY	UPLOAD A BLOB WITH A CUSTOM POLICY	UPLOAD A BLOB WITH NO POLICY
Default policy on account or container (unlocked)	Blob is uploaded with default unlocked policy	Blob is uploaded with custom unlocked policy	Blob is uploaded with no policy
Default policy on account or container (locked)	Blob is uploaded with default locked policy	Blob is uploaded with custom unlocked policy	Blob is uploaded with no policy
No default policy on either account or container	N/A	Blob is uploaded with custom unlocked policy	Blob is uploaded with no policy

#### Configure a policy on a previous version

When versioning is enabled, a write or delete operation to a blob creates a new previous version of that blob that saves the blob's state before the operation. By default, a previous version possesses the time-based retention policy that was in effect for the current version, if any, when the current version became a previous version. The new current version inherits the policy on the container, if there is one.

If the policy inherited by a previous version is unlocked, then the retention interval can be shortened or lengthened, or the policy can be deleted. The policy on a previous version can also be locked for that version, even if the policy on the current version is unlocked.

If the policy inherited by a previous version is locked, then the retention interval can be lengthened. The policy cannot be deleted, nor can the retention interval be shortened.

If there is no policy configured on the current version, then the previous version does not inherit any policy. You can configure a custom policy for the version.

If the policy on a current version is modified, the policies on existing previous versions remain unchanged, even if the policy was inherited from a current version.

#### Container-level policy scope

A container-level time-based retention policy applies to all objects in a container, both new and existing. For an account with a hierarchical namespace, a container-level policy also applies to all directories in the container.

When a time-based retention policy is applied to a container, all existing blobs move into an immutable WORM state in less than 30 seconds. All new blobs that are uploaded to that policy-protected container will also move into an immutable state. Once all blobs are in an immutable state, overwrite or delete operations in the immutable container are not allowed. In the case of an account with a hierarchical namespace, blobs cannot be renamed or moved to a different directory.

The following limits apply to container-level retention policies:

- For a storage account, the maximum number of containers with locked time-based immutable policies is 10,000.
- For a container, the maximum number of edits to extend the retention interval for a locked time-based policy is five.
- For a container, a maximum of seven time-based retention policy audit logs are retained for a locked policy.

To learn how to configure a time-based retention policy on a container, see [Configure immutability policies for containers](#).

## Allow protected append blobs writes

Append blobs are comprised of blocks of data and optimized for data append operations required by auditing and logging scenarios. By design, append blobs only allow the addition of new blocks to the end of the blob. Regardless of immutability, modification or deletion of existing blocks in an append blob is fundamentally not allowed. To learn more about append blobs, see [About Append Blobs](#).

Only time-based retention policies have an the **AllowProtectedAppendWrites** property setting that allows for writing new blocks to an append blob while maintaining immutability protection and compliance. If this setting is enabled, you can create an append blob directly in the policy-protected container and then continue to add new blocks of data to the end of the append blob with the Append Block operation. Only new blocks can be added; any existing blocks cannot be modified or deleted. Time-retention immutability protection still applies, preventing deletion of the append blob until the effective retention period has elapsed. Enabling this setting does not affect the immutability behavior of block blobs or page blobs.

As this setting is part of a time-based retention policy, the append blobs remain in the immutable state for the duration of the *effective* retention period. Since new data can be appended beyond the initial creation of the append blob, there is a slight difference in how the retention period is determined. The effective retention is the difference between append blob's last modification time and the user-specified retention interval. Similarly, when the retention interval is extended, immutable storage uses the most recent value of the user-specified retention interval to calculate the effective retention period.

For example, suppose that a user creates a time-based retention policy with the **AllowProtectedAppendWrites** property enabled and a retention interval of 90 days. An append blob, *logblob1*, is created in the container today, new logs continue to be added to the append blob for the next 10 days, so that the effective retention period for *logblob1* is 100 days from today (the time of its last append + 90 days).

Unlocked time-based retention policies allow the **AllowProtectedAppendWrites** property setting to be enabled and disabled at any time. Once the time-based retention policy is locked, the **AllowProtectedAppendWrites** property setting cannot be changed.

## Audit logging

Each container with a time-based retention policy enabled provides a policy audit log. The audit log includes up to seven time-based retention commands for locked time-based retention policies. Log entries include the user ID, command type, time stamps, and retention interval. The audit log is retained for the lifetime of the policy, in accordance with the SEC 17a-4(f) regulatory guidelines.

The [Azure Activity log](#) provides a more comprehensive log of all management service activities. [Azure resource logs](#) retain information about data operations. It is the user's responsibility to store those logs persistently, as might be required for regulatory or other purposes.

Changes to time-based retention policies at the version level are not audited.

## Next steps

- [Data protection overview](#)
- [Store business-critical blob data with immutable storage](#)
- [Legal holds for immutable blob data](#)
- [Configure immutability policies for blob versions](#)
- [Configure immutability policies for containers](#)

# Legal holds for immutable blob data

8/22/2022 • 3 minutes to read • [Edit Online](#)

A legal hold is a temporary immutability policy that can be applied for legal investigation purposes or general protection policies. A legal hold stores blob data in a Write-Once, Read-Many (WORM) format until it is explicitly cleared. When a legal hold is in effect, blobs can be created and read, but not modified or deleted. Use a legal hold when the period of time that the data must be kept in a WORM state is unknown.

For more information about immutability policies for Blob Storage, see [Store business-critical blob data with immutable storage](#).

## Legal hold scope

A legal hold policy can be configured at either of the following scopes:

- **Version-level policy:** A legal hold can be configured on an individual blob version level for granular management of sensitive data.
- **Container-level policy:** A legal hold that is configured at the container level applies to all blobs in that container. Individual blobs cannot be configured with their own immutability policies.

### Version-level policy scope

To configure a legal hold on a blob version, you must first enable version-level immutability on the storage account or the parent container. Version-level immutability cannot be disabled after it is enabled. For more information, [Enable support for version-level immutability](#).

After version-level immutability is enabled for a storage account or a container, a legal hold can no longer be set at the container level. Legal holds must be applied to individual blob versions. A legal hold may be configured for the current version or a previous version of a blob.

Version-level legal hold policies require that blob versioning is enabled for the storage account. To learn how to enable blob versioning, see [Enable and manage blob versioning](#). Keep in mind that enabling versioning may have a billing impact. For more information, see the **Pricing and billing** section in [Blob versioning](#).

To learn more about enabling a version-level legal hold, see [Configure or clear a legal hold](#).

### Container-level scope

When you configure a legal hold for a container, that hold applies to all objects in the container. When the legal hold is cleared, clients can once again write and delete objects in the container, unless there is also a time-based retention policy in effect for the container.

When a legal hold is applied to a container, all existing blobs move into an immutable WORM state in less than 30 seconds. All new blobs that are uploaded to that policy-protected container will also move into an immutable state. Once all blobs are in an immutable state, overwrite or delete operations in the immutable container are not allowed. In the case of an account with a hierarchical namespace, blobs cannot be renamed or moved to a different directory.

To learn how to configure a legal hold with container-level scope, see [Configure or clear a legal hold](#).

### Legal hold tags

A container-level legal hold must be associated with one or more user-defined alphanumeric tags that serve as identifier strings. For example, a tag may include a case ID or event name.

### Audit logging

Each container with a legal hold in effect provides a policy audit log. The log contains the user ID, command type, time stamps, and legal hold tags. The audit log is retained for the lifetime of the policy, in accordance with the SEC 17a-4(f) regulatory guidelines.

The [Azure Activity log](#) provides a more comprehensive log of all management service activities. [Azure resource logs](#) retain information about data operations. It is the user's responsibility to store those logs persistently, as might be required for regulatory or other purposes.

#### Limits

The following limits apply to container-level legal holds:

- For a storage account, the maximum number of containers with a legal hold setting is 10,000.
- For a container, the maximum number of legal hold tags is ten.
- The minimum length of a legal hold tag is three alphanumeric characters. The maximum length is 23 alphanumeric characters.
- For a container, a maximum of ten legal hold policy audit logs are retained for the duration of the policy.

## Next steps

- [Data protection overview](#)
- [Store business-critical blob data with immutable storage](#)
- [Time-based retention policies for immutable blob data](#)
- [Configure immutability policies for blob versions](#)
- [Configure immutability policies for containers](#)

# Azure Storage redundancy

8/22/2022 • 19 minutes to read • [Edit Online](#)

Azure Storage always stores multiple copies of your data so that it's 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.

When deciding which redundancy option is best for your scenario, consider the tradeoffs between lower costs and higher availability. 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 (geo-replication).
- Whether your application requires read access to the replicated data in the secondary region if the primary region becomes unavailable for any reason (geo-replication with read access).

## NOTE

The features and regional availability described in this article are also available to accounts that have a hierarchical namespace (Azure Blob storage).

The services that comprise Azure Storage are managed through a common Azure resource called a *storage account*. The storage account represents a shared pool of storage that can be used to deploy storage resources such as blob containers (Blob Storage), file shares (Azure Files), tables (Table Storage), or queues (Queue Storage). For more information about Azure Storage accounts, see [Storage account overview](#).

The redundancy setting for a storage account is shared for all storage services exposed by that account. All storage resources deployed in the same storage account have the same redundancy setting. You may want to isolate different types of resources in separate storage accounts if they have different redundancy requirements.

## 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 isn't 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.

## NOTE

Microsoft recommends using ZRS in the primary region for Azure Data Lake Storage Gen2 workloads.

### Locally redundant storage

Locally redundant storage (LRS) replicates your storage account three times within a single data center in the

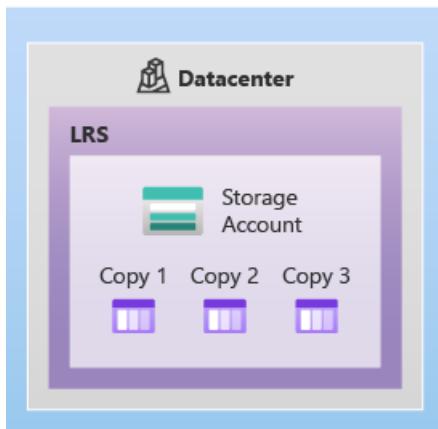
primary region. LRS provides at least 99.999999999% (11 nines) durability of objects over a given year.

LRS is the lowest-cost redundancy option and offers the least durability compared to other options. LRS protects your data against server rack and drive failures. However, if a disaster such as fire or flooding occurs within the data center, all replicas of a storage account using LRS may be lost or unrecoverable. To mitigate this risk, Microsoft recommends using [zone-redundant storage](#) (ZRS), [geo-redundant storage](#) (GRS), or [geo-zone-redundant storage](#) (GZRS).

A write request to a storage account that is using LRS happens synchronously. The write operation returns successfully only after the data is written to all three replicas.

The following diagram shows how your data is replicated within a single data center with LRS:

#### Primary region



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. In some cases, the paired regions across which the data is geo-replicated may be in another country or region. For more information on paired regions, see [Azure regions](#).
- If your scenario is using Azure unmanaged disks, you may opt for LRS. While it's possible to create a storage account for Azure unmanaged disks that uses GRS, it isn't recommended due to potential issues with consistency over asynchronous geo-replication.

#### Zone-redundant storage

Zone-redundant storage (ZRS) replicates your storage account synchronously across three Azure availability zones in the primary region. Each availability zone is a separate physical location with independent power, cooling, and networking. ZRS offers durability for storage resources of at least 99.9999999999% (12 9's) over a given year.

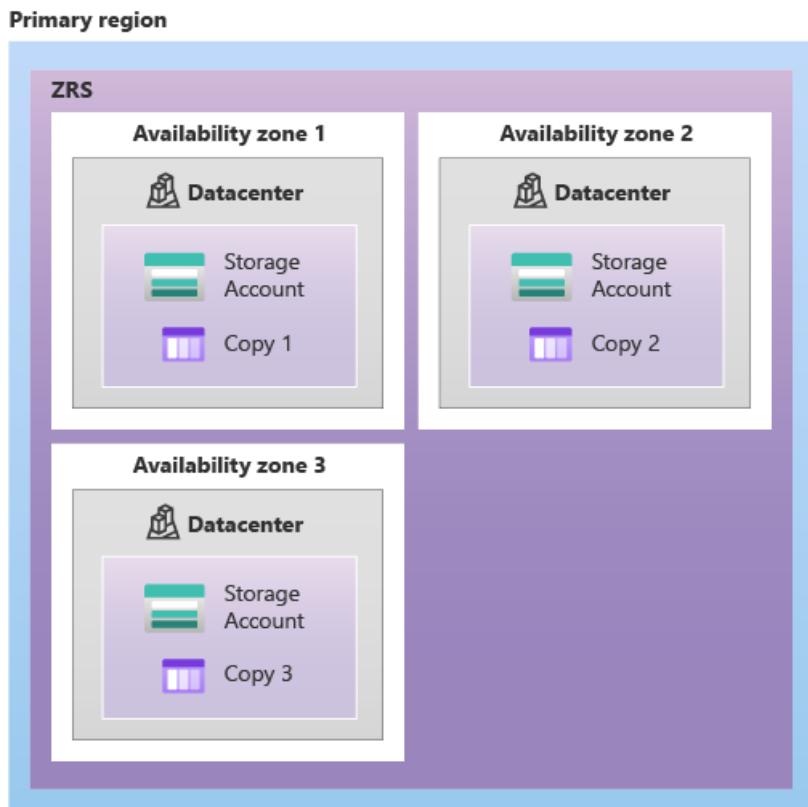
With ZRS, your data is still accessible for both read and write operations even if a zone becomes unavailable. If a zone becomes unavailable, Azure undertakes networking updates, such as DNS repointing. These updates may affect your application if you access data before the updates have completed. When designing applications for ZRS, follow practices for transient fault handling, including implementing retry policies with exponential back-off.

A write request to a storage account that is using ZRS happens synchronously. The write operation returns successfully only after the data is written to all replicas across the three availability zones.

Microsoft recommends using ZRS in the primary region for scenarios that require high availability. ZRS is also recommended for restricting replication of data to a particular country or region to meet data governance requirements.

Microsoft recommends using ZRS for Azure Files workloads. If a zone becomes unavailable, no remounting of Azure file shares from the connected clients is required.

The following diagram shows how your data is replicated across availability zones in the primary region with ZRS:



ZRS provides excellent performance, low latency, and resiliency for your data if it becomes temporarily unavailable. However, ZRS by itself may not protect your data against a regional disaster where multiple zones are permanently affected. For protection against regional disasters, Microsoft recommends using [geo-zone-redundant storage](#) (GZRS), which uses ZRS in the primary region and also geo-replicates your data to a secondary region.

The Archive tier for Blob Storage isn't currently supported for ZRS, GZRS, or RA-GZRS accounts. Unmanaged disks don't support ZRS or GZRS.

For more information about which regions support ZRS, see [Azure regions with availability zones](#).

#### Standard storage accounts

ZRS is supported for all Azure Storage services through standard general-purpose v2 storage accounts, including:

- Azure Blob storage (hot and cool block blobs and append blobs, non-disk page blobs)
- Azure Files (all standard tiers: transaction optimized, hot, and cool)
- Azure Table storage
- Azure Queue storage

ZRS for standard general-purpose v2 storage accounts is available for a subset of Azure regions:

- (Africa) South Africa North
- (Asia Pacific) Australia East
- (Asia Pacific) Central India
- (Asia Pacific) East Asia
- (Asia Pacific) Japan East
- (Asia Pacific) Korea Central
- (Asia Pacific) South India
- (Asia Pacific) Southeast Asia

- (Europe) France Central
- (Europe) Germany West Central
- (Europe) North Europe
- (Europe) Norway East
- (Europe) Sweden Central
- (Europe) Switzerland North
- (Europe) UK South
- (Europe) West Europe
- (North America) Canada Central
- (North America) Central US
- (North America) East US
- (North America) East US 2
- (North America) South Central US
- (North America) US Gov Virginia
- (North America) West US 2
- (North America) West US 3
- (South America) Brazil South

#### Premium block blob accounts

ZRS is supported for premium block blobs accounts. For more information about premium block blobs, see [Premium block blob storage accounts](#).

Premium block blobs are available in a subset of Azure regions:

- (Asia Pacific) Australia East
- (Asia Pacific) East Asia
- (Asia Pacific) Japan East
- (Asia Pacific) Southeast Asia
- (Europe) France Central
- (Europe) North Europe
- (Europe) West Europe
- (Europe) UK South
- (North America) East US
- (North America) East US 2
- (North America) West US 2
- (North America) South Central US
- (South America) Brazil South

#### Premium file share accounts

ZRS is supported for premium file shares (Azure Files) through the `FileStorage` storage account kind.

ZRS for premium file shares is available for a subset of Azure regions:

- (Asia Pacific) Australia East
- (Asia Pacific) Japan East
- (Asia Pacific) Southeast Asia
- (Europe) France Central
- (Europe) North Europe
- (Europe) West Europe
- (Europe) UK South

- (North America) East US
- (North America) East US 2
- (North America) West US 2
- (North America) South Central US
- (South America) Brazil South

## 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. For more information about regions supported by Azure, see [Azure regions](#).

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.

### NOTE

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.

With GRS or GZRS, the data in the secondary region isn't available for read or write access unless there's a failover to the secondary region. 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](#).

If the primary region becomes unavailable, you can choose to fail over to the secondary region. After the failover has completed, the secondary region becomes the primary region, and you can again read and write data. For more information on disaster recovery and to learn how to fail over to the secondary region, see [Disaster recovery and storage account failover](#).

### IMPORTANT

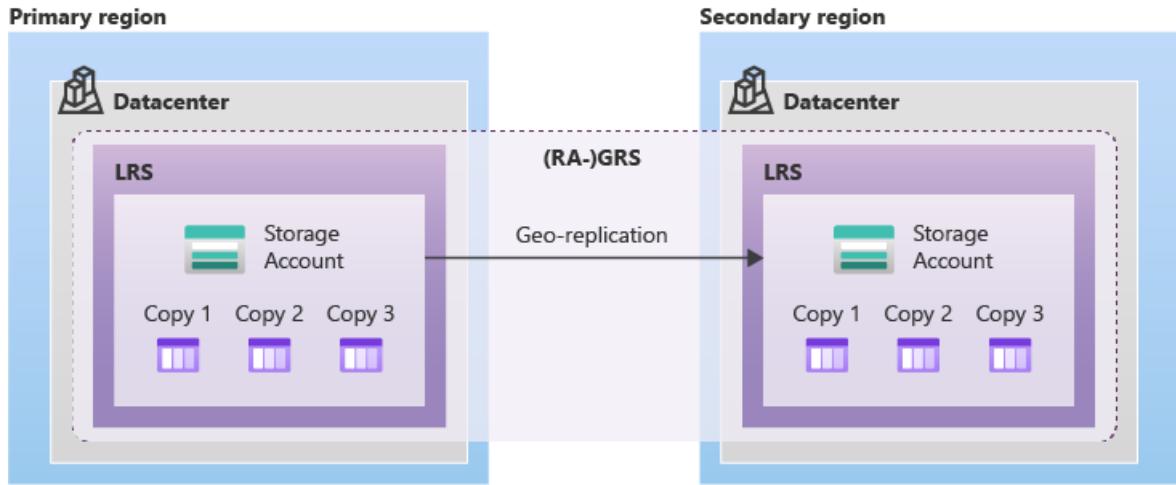
Because data is replicated to the secondary region asynchronously, a failure that affects the primary region may result in data loss if the primary region cannot be recovered. The interval between the most recent writes to the primary region and the last write to the secondary region is known as the recovery point objective (RPO). The RPO indicates the point in time to which data can be recovered. The Azure Storage platform typically has an RPO of less than 15 minutes, although there's currently no SLA on how long it takes to replicate data to the secondary region.

## Geo-redundant storage

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 a secondary region that is hundreds of miles away from the primary region. GRS offers durability for storage resources of at least 99.999999999999% (16 9's) over a given year.

A write operation is first committed to the primary location and replicated using LRS. The update is then replicated asynchronously to the secondary region. When data is written to the secondary location, it's also replicated within that location using LRS.

The following diagram shows how your data is replicated with GRS or RA-GRS:

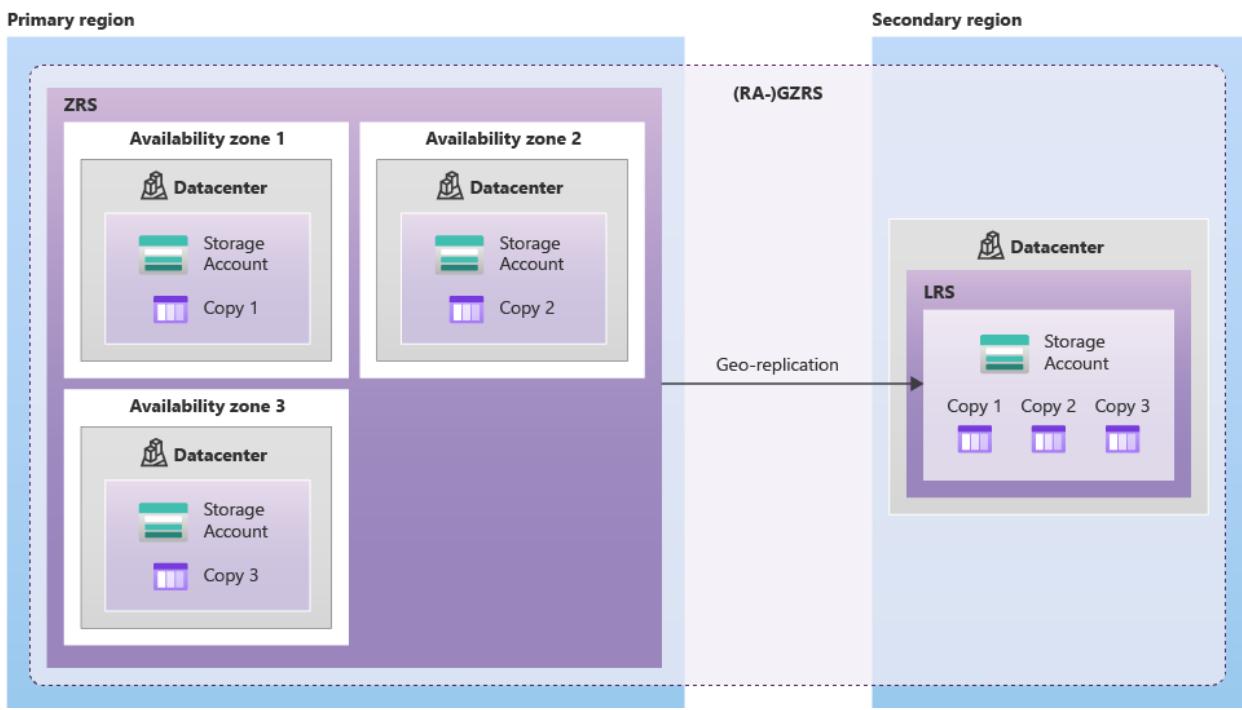


### Geo-zone-redundant storage

Geo-zone-redundant storage (GZRS) combines the high availability provided by redundancy across availability zones with protection from regional outages provided by geo-replication. Data in a GZRS storage account is copied across three [Azure availability zones](#) in the primary region and is also replicated to a secondary geographic region for protection from regional disasters. Microsoft recommends using GZRS for applications requiring maximum consistency, durability, and availability, excellent performance, and resilience for disaster recovery.

With a GZRS storage account, you can continue to read and write data if an availability zone becomes unavailable or is unrecoverable. Additionally, your data is also durable in the case of a complete regional outage or a disaster in which the primary region isn't recoverable. GZRS is designed to provide at least 99.999999999999% (16 9's) durability of objects over a given year.

The following diagram shows how your data is replicated with GZRS or RA-GZRS:



Only standard general-purpose v2 storage accounts support GZRS. GZRS is supported by all of the Azure Storage services, including:

- Azure Blob storage (hot and cool block blobs, non-disk page blobs)
- Azure Files (all standard tiers: transaction optimized, hot, and cool)
- Azure Table storage
- Azure Queue storage

GZRS is available for a subset of Azure regions:

- (Africa) South Africa North
- (Asia Pacific) Australia East
- (Asia Pacific) East Asia
- (Asia Pacific) Japan East
- (Asia Pacific) Korea Central
- (Asia Pacific) Southeast Asia
- (Asia Pacific) Central India
- (Europe) France Central
- (Europe) North Europe
- (Europe) Norway East
- (Europe) Sweden Central
- (Europe) Switzerland North
- (Europe) UK South
- (Europe) West Europe
- (North America) Canada Central
- (North America) Central US
- (North America) East US
- (North America) East US 2
- (North America) South Central US
- (North America) West US 2
- (North America) West US 3
- (North America) US Gov Virginia

- (South America) Brazil South

## Read access to data in the secondary region

Geo-redundant storage (with GRS or GZRS) replicates your data to another physical location in the secondary region to protect against regional outages. With an account configured for GRS or GZRS, data in the secondary region is not directly accessible to users or applications, unless a failover occurs. The failover process updates the DNS entry provided by Azure Storage so that the secondary endpoint becomes the new primary endpoint for your storage account. During the failover process, your data is inaccessible. After the failover is complete, you can read and write data to the new primary region. For more information about failover and disaster recovery, see [How an account failover works](#).

If your applications require high availability, then you can configure your storage account for read access to the secondary region. When you enable read access to the secondary region, then your data is always available to be read from the secondary, including in a situation where the primary region becomes unavailable. Read-access geo-redundant storage (RA-GRS) or read-access geo-zone-redundant storage (RA-GZRS) configurations permit read access to the secondary region.

**Caution**

Because data is replicated asynchronously from the primary to the secondary region, the secondary region is typically behind the primary region in terms of write operations. If a disaster were to strike the primary region, it's likely that some data would be lost. For more information about how to plan for potential data loss, see [Anticipate data loss](#).

**NOTE**

Azure Files does not support read-access geo-redundant storage (RA-GRS) or read-access geo-zone-redundant storage (RA-GZRS).

### Design your applications for read access to the secondary

If your storage account is configured for read access to the secondary region, then you can design your applications to seamlessly shift to reading data from the secondary region if the primary region becomes unavailable for any reason.

The secondary region is available for read access after you enable RA-GRS or RA-GZRS, so that you can test your application in advance to make sure that it will properly read from the secondary in the event of an outage. For more information about how to design your applications to take advantage of geo-redundancy, see [Use geo-redundancy to design highly available applications](#).

When read access to the secondary is enabled, your application can be read from the secondary endpoint as well as from the primary endpoint. The secondary endpoint appends the suffix `-secondary` to the account name. For example, if your primary endpoint for Blob storage is `myaccount.blob.core.windows.net`, then the secondary endpoint is `myaccount-secondary.blob.core.windows.net`. The account access keys for your storage account are the same for both the primary and secondary endpoints.

### Check the Last Sync Time property

Because data is replicated to the secondary region asynchronously, the secondary region is often behind the primary region. If a failure happens in the primary region, it's likely that all writes to the primary won't yet have been replicated to the secondary.

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're available to be read from the secondary. Any write operations written to the primary region after the last sync time may or may not have been replicated to the secondary region, meaning that they may not be available for read

operations.

You can query the value of the **Last Sync Time** property using Azure PowerShell, Azure CLI, or one of the Azure Storage client libraries. The **Last Sync Time** property is a GMT date/time value. For more information, see [Check the Last Sync Time property for a storage account](#).

## Summary of redundancy options

The tables in the following sections summarize the redundancy options available for Azure Storage.

### Durability and availability parameters

The following table describes key parameters for each redundancy option:

PARAMETER	LRS	ZRS	GRS/RA-GRS	GZRS/RA-GZRS
Percent durability of objects over a given year	at least 99.99999999% (11 9's)	at least 99.999999999% (12 9's)	at least 99.99999999999999% (16 9's)	at least 99.99999999999999% (16 9's)
Availability for read requests	At least 99.9% (99% for Cool or Archive access tiers)	At least 99.9% (99% for Cool or Archive access tiers)	At least 99.9% (99% for Cool or Archive access tiers) for GRS At least 99.99% (99.9% for Cool or Archive access tiers) for RA-GRS	At least 99.9% (99% for Cool or Archive access tiers) for GZRS At least 99.99% (99.9% for Cool or Archive access tiers) for RA-GZRS
Availability for write requests	At least 99.9% (99% for Cool or Archive access tiers)	At least 99.9% (99% for Cool or Archive access tiers)	At least 99.9% (99% for Cool or Archive access tiers)	At least 99.9% (99% for Cool or Archive access tiers)
Number of copies of data maintained on separate nodes	Three copies within a single region	Three copies across separate availability zones within a single region	Six copies total, including three in the primary region and three in the secondary region	Six copies total, including three across separate availability zones in the primary region and three locally redundant copies in the secondary region

For more information, see the [SLA for Storage Accounts](#).

### Durability and availability by outage scenario

The following table indicates whether your data is durable and available in a given scenario, depending on which type of redundancy is in effect for your storage account:

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	Yes <sup>1</sup>	Yes

OUTAGE SCENARIO	LRS	ZRS	GRS/RA-GRS	GZRS/RA-GZRS
A region-wide outage occurs in the primary region	No	No	Yes <sup>1</sup>	Yes <sup>1</sup>
Read access to the secondary region is available if the primary region becomes unavailable	No	No	Yes (with RA-GRS)	Yes (with RA-GZRS)

<sup>1</sup> Account failover is required to restore write availability if the primary region becomes unavailable. For more information, see [Disaster recovery and storage account failover](#).

## Supported Azure Storage services

The following table shows which redundancy options are supported by each Azure Storage service.

LRS	ZRS	GRS	RA-GRS	GZRS	RA-GZRS
Blob storage (including Data Lake Storage)					
Queue storage					
Table storage					
Azure Files <sup>1,2</sup>	Azure Files <sup>1,2</sup>	Azure Files <sup>1</sup>			
Azure managed disks	Azure managed disks <sup>3</sup>				
Page blobs					

<sup>1</sup> Standard file shares are supported on LRS and ZRS. Standard file shares are supported on GRS and GZRS as long as they're less than or equal to 5 TiB in size.

<sup>2</sup> Premium file shares are supported on LRS and ZRS.

<sup>3</sup> ZRS managed disks have certain limitations. See the [Limitations](#) section of the redundancy options for managed disks article for details.

## Supported storage account types

The following table shows which redundancy options are supported for each type of storage account. For information for storage account types, see [Storage account overview](#).

STORAGE ACCOUNT TYPES	LRS	ZRS	GRS/RA-GRS	GZRS/RA-GZRS
Recommended	Standard general-purpose v2 ( <code>StorageV2</code> ) <sup>1</sup>  Premium block blobs ( <code>BlockBlobStorage</code> ) <sup>1</sup>  Premium file shares ( <code>FileStorage</code> )  Premium page blobs ( <code>StorageV2</code> )	Standard general-purpose v2 ( <code>StorageV2</code> ) <sup>1</sup>  Premium block blobs ( <code>BlockBlobStorage</code> ) <sup>1</sup>  Premium file shares ( <code>FileStorage</code> )	Standard general-purpose v2 ( <code>StorageV2</code> ) <sup>1</sup>	Standard general-purpose v2 ( <code>StorageV2</code> ) <sup>1</sup>

STORAGE ACCOUNT TYPES	LRS	ZRS	GRS/RA-GRS	GZRS/RA-GZRS
Legacy	Standard general-purpose v1 ( Storage)  Legacy blob ( BlobStorage)	N/A	Standard general-purpose v1 ( Storage)  Legacy blob ( BlobStorage)	N/A

<sup>1</sup> Accounts of this type with a hierarchical namespace enabled also support the specified redundancy option.

All data for all storage accounts is copied from the primary to the secondary according to the redundancy option for the storage account. Objects including block blobs, append blobs, page blobs, queues, tables, and files are copied.

Data in all tiers, including the Archive tier, is always copied from the primary to the secondary during geo-replication. The Archive tier for Blob Storage is currently supported for LRS, GRS, and RA-GRS accounts, but not for ZRS, GZRS, or RA-GZRS accounts. For more information about blob tiers, see [Hot, Cool, and Archive access tiers for blob data](#).

Unmanaged disks don't support ZRS or GZRS.

For pricing information for each redundancy option, see [Azure Storage pricing](#).

#### NOTE

Azure Premium Disk Storage currently supports only locally redundant storage (LRS). Block blob storage accounts support locally redundant storage (LRS) and zone redundant storage (ZRS) in certain regions.

## Support for customer-managed account failover

All geo-redundant offerings support Microsoft-managed failover in the event of a disaster in the primary region. In addition, some account types support customer-managed account failover, as shown in the following table. Supported account types must use Azure Resource Manager deployments. For more information about disaster recovery and customer-managed failover, see [Disaster recovery and storage account failover](#).

TYPE OF FAILOVER	GRS/RA-GRS	GZRS/RA-GZRS
Customer-managed failover	General-purpose v2 accounts General-purpose v1 accounts Legacy Blob Storage accounts	General-purpose v2 accounts
Microsoft-managed failover	All account types	General-purpose v2 accounts

#### NOTE

Customer-managed account failover is not yet supported in accounts that have a hierarchical namespace (Azure Data Lake Storage Gen2). To learn more, see [Blob storage features available in Azure Data Lake Storage Gen2](#).

In the event of a disaster that affects the primary region, Microsoft will manage the failover for accounts with a hierarchical namespace. For more information, see [Microsoft-managed failover](#).

## Data integrity

Azure Storage regularly verifies the integrity of data stored using cyclic redundancy checks (CRCs). If data

corruption is detected, it's repaired using redundant data. Azure Storage also calculates checksums on all network traffic to detect corruption of data packets when storing or retrieving data.

## See also

- [Change the redundancy option for a storage account](#)
- Geo replication (GRS/GZRS/RA-GRS/RA-GZRS)
  - [Check the Last Sync Time property for a storage account](#)
  - [Disaster recovery and storage account failover](#)
- Pricing
  - [Blob Storage](#)
  - [Azure Files](#)
  - [Table Storage](#)
  - [Queue Storage](#)

# Disaster recovery and storage account failover

8/22/2022 • 13 minutes to read • [Edit Online](#)

Microsoft strives to ensure that Azure services are always available. However, unplanned service outages may occur. If your application requires resiliency, Microsoft recommends using geo-redundant storage, so that your data is copied to a second region. Additionally, customers should have a disaster recovery plan in place for handling a regional service outage. An important part of a disaster recovery plan is preparing to fail over to the secondary endpoint in the event that the primary endpoint becomes unavailable.

Azure Storage supports account failover for geo-redundant storage accounts. With account failover, you can initiate the failover process for your storage account if the primary endpoint becomes unavailable. The failover updates the secondary endpoint to become the primary endpoint for your storage account. Once the failover is complete, clients can begin writing to the new primary endpoint.

Account failover is available for general-purpose v1, general-purpose v2, and Blob storage account types with Azure Resource Manager deployments. Account failover is not supported for storage accounts with a hierarchical namespace enabled.

This article describes the concepts and process involved with an account failover and discusses how to prepare your storage account for recovery with the least amount of customer impact. To learn how to initiate an account failover in the Azure portal or PowerShell, see [Initiate an account failover](#).

## NOTE

This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

## Choose the right redundancy option

Azure Storage maintains multiple copies of your storage account to ensure durability and high availability. Which redundancy option you choose for your account depends on the degree of resiliency you need. For protection against regional outages, configure your account for geo-redundant storage, with or without the option of read access from the secondary region:

**Geo-redundant storage (GRS) or geo-zone-redundant storage (GZRS)** copies your data asynchronously in two geographic regions that are at least hundreds of miles apart. If the primary region suffers an outage, then the secondary region serves as a redundant source for your data. You can initiate a failover to transform the secondary endpoint into the primary endpoint.

**Read-access geo-redundant storage (RA-GRS) or read-access geo-zone-redundant storage (RA-GZRS)** provides geo-redundant storage with the additional benefit of read access to the secondary endpoint. If an outage occurs in the primary endpoint, applications configured for read access to the secondary and designed for high availability can continue to read from the secondary endpoint. Microsoft recommends RA-GZRS for maximum availability and durability for your applications.

For more information about redundancy in Azure Storage, see [Azure Storage redundancy](#).

## WARNING

Geo-redundant storage carries a risk of data loss. Data is copied to the secondary region asynchronously, meaning there is a delay between when data written to the primary region is written to the secondary region. In the event of an outage, write operations to the primary endpoint that have not yet been copied to the secondary endpoint will be lost.

## Design for high availability

It's important to design your application for high availability from the start. Refer to these Azure resources for guidance in designing your application and planning for disaster recovery:

- [Designing resilient applications for Azure](#): An overview of the key concepts for architecting highly available applications in Azure.
- [Resiliency checklist](#): A checklist for verifying that your application implements the best design practices for high availability.
- [Use geo-redundancy to design highly available applications](#): Design guidance for building applications to take advantage of geo-redundant storage.
- [Tutorial: Build a highly available application with Blob storage](#): A tutorial that shows how to build a highly available application that automatically switches between endpoints as failures and recoveries are simulated.

Additionally, keep in mind these best practices for maintaining high availability for your Azure Storage data:

- **Disks:** Use [Azure Backup](#) to back up the VM disks used by your Azure virtual machines. Also consider using [Azure Site Recovery](#) to protect your VMs in the event of a regional disaster.
- **Block blobs:** Turn on [soft delete](#) to protect against object-level deletions and overwrites, or copy block blobs to another storage account in a different region using [AzCopy](#), [Azure PowerShell](#), or the [Azure Data Movement library](#).
- **Files:** Use [Azure Backup](#) to back up your file shares. Also enable [soft delete](#) to protect against accidental file share deletions. For geo-redundancy when GRS is not available, use [AzCopy](#) or [Azure PowerShell](#) to copy your files to another storage account in a different region.
- **Tables:** use [AzCopy](#) to export table data to another storage account in a different region.

## Track outages

Customers may subscribe to the [Azure Service Health Dashboard](#) to track the health and status of Azure Storage and other Azure services.

Microsoft also recommends that you design your application to prepare for the possibility of write failures. Your application should expose write failures in a way that alerts you to the possibility of an outage in the primary region.

## Understand the account failover process

Customer-managed account failover enables you to fail your entire storage account over to the secondary region if the primary becomes unavailable for any reason. When you force a failover to the secondary region, clients can begin writing data to the secondary endpoint after the failover is complete. The failover typically takes about an hour.

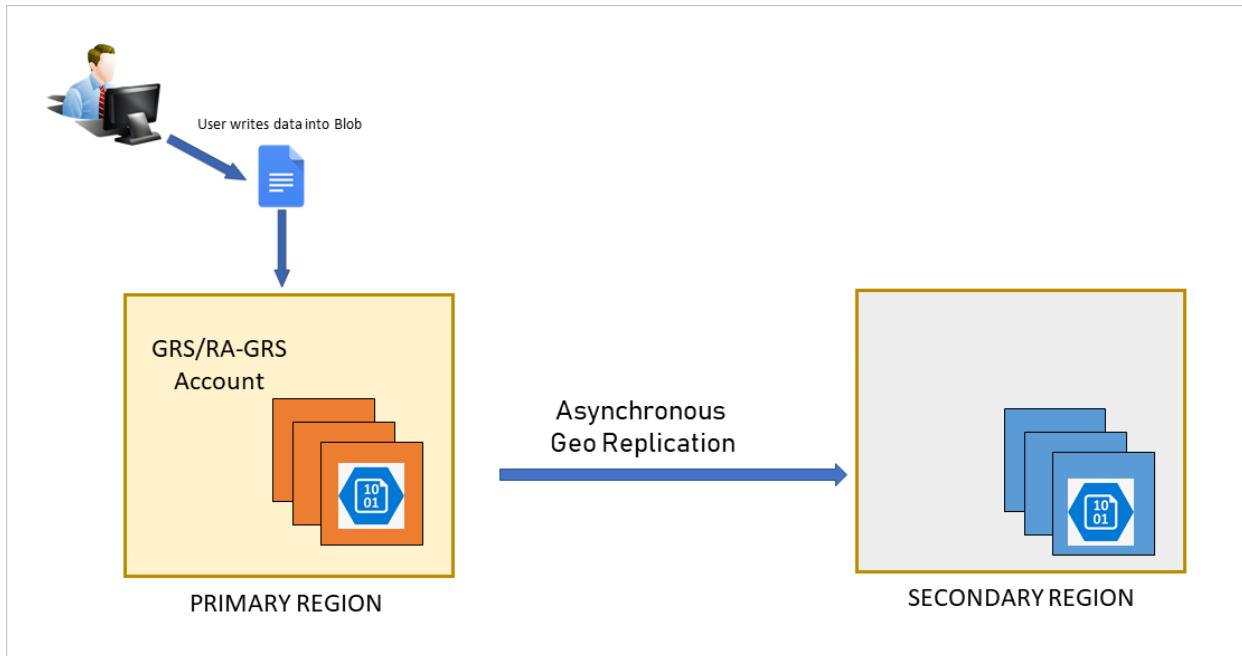
## NOTE

Customer-managed account failover is not yet supported in accounts that have a hierarchical namespace (Azure Data Lake Storage Gen2). To learn more, see [Blob storage features available in Azure Data Lake Storage Gen2](#).

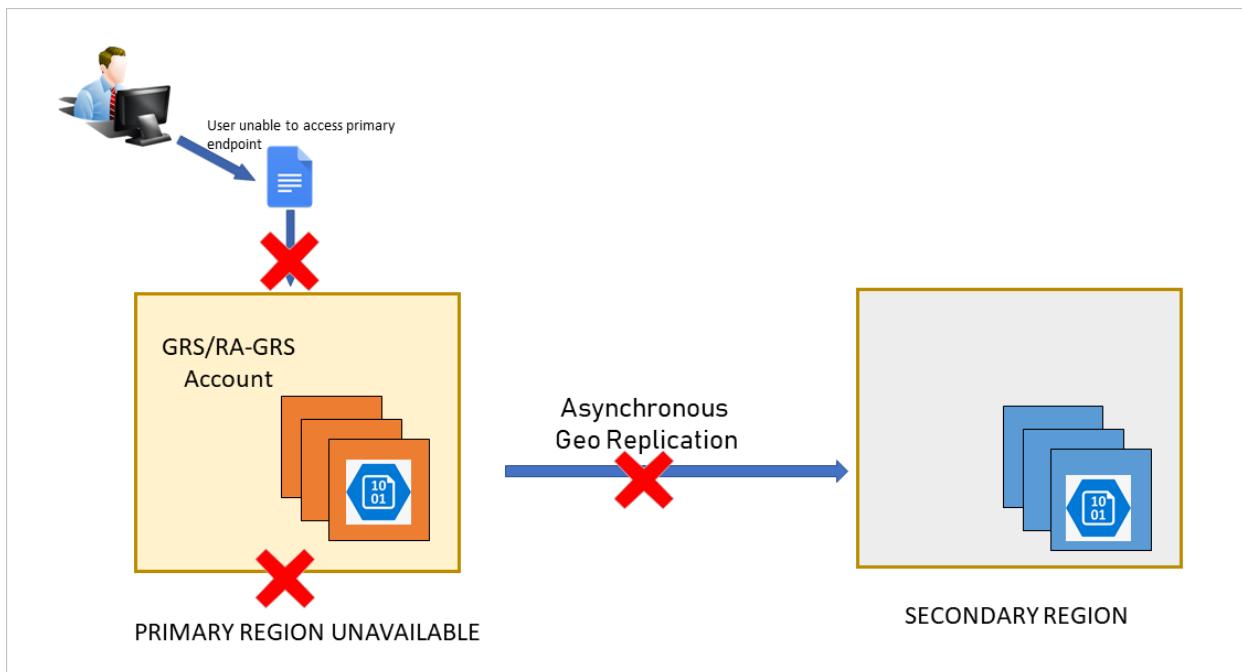
In the event of a disaster that affects the primary region, Microsoft will manage the failover for accounts with a hierarchical namespace. For more information, see [Microsoft-managed failover](#).

## How an account failover works

Under normal circumstances, a client writes data to an Azure Storage account in the primary region, and that data is copied asynchronously to the secondary region. The following image shows the scenario when the primary region is available:

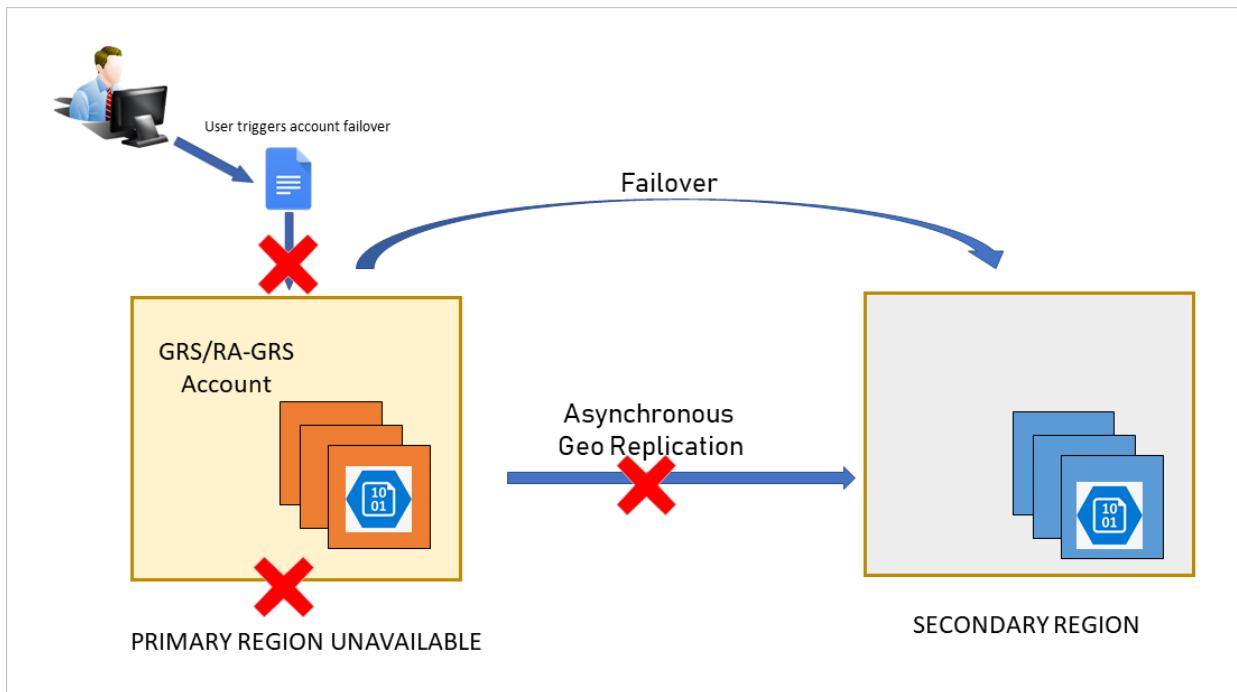


If the primary endpoint becomes unavailable for any reason, the client is no longer able to write to the storage account. The following image shows the scenario where the primary has become unavailable, but no recovery has happened yet:



The customer initiates the account failover to the secondary endpoint. The failover process updates the DNS

entry provided by Azure Storage so that the secondary endpoint becomes the new primary endpoint for your storage account, as shown in the following image:



Write access is restored for geo-redundant accounts once the DNS entry has been updated and requests are being directed to the new primary endpoint. Existing storage service endpoints for blobs, tables, queues, and files remain the same after the failover.

#### IMPORTANT

After the failover is complete, the storage account is configured to be locally redundant in the new primary endpoint. To resume replication to the new secondary, configure the account for geo-redundancy again.

Keep in mind that converting a locally redundant storage account to use geo-redundancy incurs both cost and time. For more information, see [Important implications of account failover](#).

#### Anticipate data loss

##### Caution

An account failover usually involves some data loss. It's important to understand the implications of initiating an account failover.

Because data is written asynchronously from the primary region to the secondary region, there is always a delay before a write to the primary region is copied to the secondary region. If the primary region becomes unavailable, the most recent writes may not yet have been copied to the secondary region.

When you force a failover, all data in the primary region is lost as the secondary region becomes the new primary region. The new primary region is configured to be locally redundant after the failover.

All data already copied to the secondary is maintained when the failover happens. However, any data written to the primary that has not also been copied to the secondary is lost permanently.

The **Last Sync Time** property indicates the most recent time that data from the primary region is guaranteed to have been written to the secondary region. All data written prior to the last sync time is available on the secondary, while data written after the last sync time may not have been written to the secondary and may be lost. Use this property in the event of an outage to estimate the amount of data loss you may incur by initiating an account failover.

As a best practice, design your application so that you can use the last sync time to evaluate expected data loss.

For example, if you are logging all write operations, then you can compare the time of your last write operations to the last sync time to determine which writes have not been synced to the secondary.

For more information about checking the **Last Sync Time** property, see [Check the Last Sync Time property for a storage account](#).

### Use caution when failing back to the original primary

After you fail over from the primary to the secondary region, your storage account is configured to be locally redundant in the new primary region. You can then configure the account in the new primary region for geo-redundancy. When the account is configured for geo-redundancy after a failover, the new primary region immediately begins copying data to the new secondary region, which was the primary before the original failover. However, it may take some time before existing data in the new primary is fully copied to the new secondary.

After the storage account is reconfigured for geo-redundancy, it's possible to initiate a failback from the new primary to the new secondary. In this case, the original primary region prior to the failover becomes the primary region again, and is configured to be either locally redundant or zone-redundant, depending on whether the original primary configuration was GRS/RA-GRS or GZRS/RA-GZRS. All data in the post-failover primary region (the original secondary) is lost during the failback. If most of the data in the storage account has not been copied to the new secondary before you fail back, you could suffer a major data loss.

To avoid a major data loss, check the value of the **Last Sync Time** property before failing back. Compare the last sync time to the last times that data was written to the new primary to evaluate expected data loss.

After a failback operation, you can configure the new primary region to be geo-redundant again. If the original primary was configured for LRS, you can configure it to be GRS or RA-GRS. If the original primary was configured for ZRS, you can configure it to be GZRS or RA-GZRS. For additional options, see [Change how a storage account is replicated](#).

## Initiate an account failover

You can initiate an account failover from the Azure portal, PowerShell, Azure CLI, or the Azure Storage resource provider API. For more information on how to initiate a failover, see [Initiate an account failover](#).

## Additional considerations

Review the additional considerations described in this section to understand how your applications and services may be affected when you force a failover.

### Storage account containing archived blobs

Storage accounts containing archived blobs support account failover. After failover is complete, all archived blobs need to be rehydrated to an online tier before the account can be configured for geo-redundancy.

### Storage resource provider

Microsoft provides two REST APIs for working with Azure Storage resources. These APIs form the basis of all actions you can perform against Azure Storage. The Azure Storage REST API enables you to work with data in your storage account, including blob, queue, file, and table data. The Azure Storage resource provider REST API enables you to manage the storage account and related resources.

After a failover is complete, clients can again read and write Azure Storage data in the new primary region. However, the Azure Storage resource provider does not fail over, so resource management operations must still take place in the primary region. If the primary region is unavailable, you will not be able to perform management operations on the storage account.

Because the Azure Storage resource provider does not fail over, the **Location** property will return the original primary location after the failover is complete.

## Azure virtual machines

Azure virtual machines (VMs) do not fail over as part of an account failover. If the primary region becomes unavailable, and you fail over to the secondary region, then you will need to recreate any VMs after the failover. Also, there is a potential data loss associated with the account failover. Microsoft recommends the following [high availability](#) and [disaster recovery](#) guidance specific to virtual machines in Azure.

## Azure unmanaged disks

As a best practice, Microsoft recommends converting unmanaged disks to managed disks. However, if you need to fail over an account that contains unmanaged disks attached to Azure VMs, you will need to shut down the VM before initiating the failover.

Unmanaged disks are stored as page blobs in Azure Storage. When a VM is running in Azure, any unmanaged disks attached to the VM are leased. An account failover cannot proceed when there is a lease on a blob. To perform the failover, follow these steps:

1. Before you begin, note the names of any unmanaged disks, their logical unit numbers (LUN), and the VM to which they are attached. Doing so will make it easier to reattach the disks after the failover.
2. Shut down the VM.
3. Delete the VM, but retain the VHD files for the unmanaged disks. Note the time at which you deleted the VM.
4. Wait until the **Last Sync Time** has updated, and is later than the time at which you deleted the VM. This step is important, because if the secondary endpoint has not been fully updated with the VHD files when the failover occurs, then the VM may not function properly in the new primary region.
5. Initiate the account failover.
6. Wait until the account failover is complete and the secondary region has become the new primary region.
7. Create a VM in the new primary region and reattach the VHDs.
8. Start the new VM.

Keep in mind that any data stored in a temporary disk is lost when the VM is shut down.

## Unsupported features and services

The following features and services are not supported for account failover:

- Azure File Sync does not support storage account failover. Storage accounts containing Azure file shares being used as cloud endpoints in Azure File Sync should not be failed over. Doing so will cause sync to stop working and may also cause unexpected data loss in the case of newly tiered files.
- Storage accounts that have hierarchical namespace enabled (such as for Data Lake Storage Gen2) are not supported at this time.
- A storage account containing premium block blobs cannot be failed over. Storage accounts that support premium block blobs do not currently support geo-redundancy.
- A storage account containing any [WORM immutability policy](#) enabled containers cannot be failed over. Unlocked/locked time-based retention or legal hold policies prevent failover in order to maintain compliance.

## Copying data as an alternative to failover

If your storage account is configured for read access to the secondary, then you can design your application to read from the secondary endpoint. If you prefer not to fail over in the event of an outage in the primary region, you can use tools such as [AzCopy](#), [Azure PowerShell](#), or the [Azure Data Movement library](#) to copy data from your storage account in the secondary region to another storage account in an unaffected region. You can then point your applications to that storage account for both read and write availability.

### Caution

An account failover should not be used as part of your data migration strategy.

## Microsoft-managed failover

In extreme circumstances where a region is lost due to a significant disaster, Microsoft may initiate a regional failover. In this case, no action on your part is required. Until the Microsoft-managed failover has completed, you won't have write access to your storage account. Your applications can read from the secondary region if your storage account is configured for RA-GRS or RA-GZRS.

## See also

- [Use geo-redundancy to design highly available applications](#)
- [Initiate an account failover](#)
- [Check the Last Sync Time property for a storage account](#)
- [Tutorial: Build a highly available application with Blob storage](#)

# Hot, Cool, and Archive access tiers for blob data

8/22/2022 • 13 minutes to read • [Edit Online](#)

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 offers different access tiers so that you can store your blob data in the most cost-effective manner based on how it's being used. 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.
- **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.
- **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.

Azure storage capacity limits are set at the account level, rather than according to access tier. You can choose to maximize your capacity usage in one tier, or to distribute capacity across two or more tiers.

## NOTE

Setting the access tier is only allowed on Block Blobs. They are not supported for Append and Page Blobs.

## Online access tiers

When your data is stored in an online access tier (either Hot or Cool), users can access it immediately. The Hot tier is the best choice for data that is in active use, while the Cool tier is ideal for data that is accessed less frequently, but that still must be available for reading and writing.

Example usage scenarios for the Hot tier include:

- Data that's in active use or is expected to be read from and written to frequently.
- Data that's staged for processing and eventual migration to the Cool access tier.

Usage scenarios for the Cool access tier 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 additional data is being gathered for processing.

To learn how to move a blob to the Hot or Cool tier, see [Set a blob's access tier](#).

Data in the Cool tier has slightly lower availability, but offers the same high durability, retrieval latency, and throughput characteristics as the Hot tier. For data in the Cool tier, slightly lower availability and higher access costs may be acceptable trade-offs for lower overall storage costs, as compared to the Hot tier. For more information, see [SLA for storage](#).

A blob in the Cool tier in a general-purpose v2 account is subject to an early deletion penalty if it's deleted or moved to a different tier before 30 days has elapsed. This charge is prorated. For example, if a blob is moved to the Cool tier and then deleted after 21 days, you'll be charged an early deletion fee equivalent to 9 (30 minus

21) days of storing that blob in the Cool tier.

The Hot and Cool tiers support all redundancy configurations. For more information about data redundancy options in Azure Storage, see [Azure Storage redundancy](#).

## Archive access tier

The Archive tier is an offline tier for storing data that is rarely accessed. The Archive access tier has the lowest storage cost, but higher data retrieval costs and latency compared to the Hot and Cool tiers. 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

To learn how to move a blob to the Archive tier, see [Archive a blob](#).

Data must remain in the Archive tier for at least 180 days or be subject to an early deletion charge. For example, if a blob is moved to the Archive tier and then deleted or moved to the Hot tier after 45 days, you'll be charged an early deletion fee equivalent to 135 (180 minus 45) days of storing that blob in the Archive tier.

While a blob is in the Archive tier, it can't be read or modified. To read or download a blob in the Archive tier, you must first rehydrate it to an online tier, either Hot or Cool. Data in the Archive tier can take up to 15 hours to rehydrate, depending on the priority you specify for the rehydration operation. For more information about blob rehydration, see [Overview of blob rehydration from the Archive tier](#).

An archived blob's metadata remains available for read access, so that you can list the blob and its properties, metadata, and index tags. Metadata for a blob in the Archive tier is read-only, while blob index tags can be read or written. 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. For more information about redundancy configurations for Azure Storage, see [Azure Storage redundancy](#).

To change the redundancy configuration for a storage account that contains blobs in the Archive tier, you must first rehydrate all archived blobs to the Hot or Cool tier. Microsoft recommends that you avoid changing the redundancy configuration for a storage account that contains archived blobs if at all possible, because rehydration operations can be costly and time-consuming.

Migrating a storage account from LRS to GRS is supported as long as no blobs were moved to the Archive tier while the account was configured for LRS. An account can be moved back to GRS if the update is performed less than 30 days from the time the account became LRS, and no blobs were moved to the Archive tier while the account was set to LRS.

# Default account access tier setting

Storage accounts have a default access tier setting that indicates the online tier in which a new blob is created. The default access tier setting can be set to either Hot or Cool. Users can override the default setting for an individual blob when uploading the blob or changing its tier.

The default access tier for a new general-purpose v2 storage account is set to the Hot tier by default. You can change the default access tier setting when you create a storage account or after it's created. If you don't change this setting on the storage account or explicitly set the tier when uploading a blob, then a new blob is uploaded to the Hot tier by default.

A blob that doesn't have an explicitly assigned tier infers its tier from the default account access tier setting. If a blob's access tier is inferred from the default account access tier setting, then the Azure portal displays the access tier as **Hot (inferred)** or **Cool (inferred)**.

Changing the default access tier setting for a storage account applies to all blobs in the account for which an access tier hasn't been explicitly set. If you toggle the default access tier setting from Hot to Cool in a general-purpose v2 account, then you're charged for write operations (per 10,000) for all blobs for which the access tier is inferred. You're charged for both read operations (per 10,000) and data retrieval (per GB) if you toggle from Cool to Hot in a general-purpose v2 account.

When you create a legacy Blob Storage account, you must specify the default access tier setting as Hot or Cool at create time. There's no charge for changing the default account access tier setting from Hot to Cool in a legacy Blob Storage account. You're charged for both read operations (per 10,000) and data retrieval (per GB) if you toggle from Cool to Hot in a Blob Storage account. Microsoft recommends using general-purpose v2 storage accounts rather than Blob Storage accounts when possible.

## NOTE

The Archive tier is not supported as the default access tier for a storage account.

## Setting or changing a blob's tier

To explicitly set a blob's tier when you create it, specify the tier when you upload the blob.

After a blob is created, you can change its tier in either of the following ways:

- By calling the [Set Blob Tier](#) operation, either directly or via a [lifecycle management](#) policy. Calling [Set Blob Tier](#) is typically the best option when you're changing a blob's tier from a hotter tier to a cooler one.
- By calling the [Copy Blob](#) operation to copy a blob from one tier to another. Calling [Copy Blob](#) is recommended for most scenarios where you're rehydrating a blob from the Archive tier to an online tier, or moving a blob from Cool to Hot. By copying a blob, you can avoid the early deletion penalty, if the required storage interval for the source blob hasn't yet elapsed. However, copying a blob results in capacity charges for two blobs, the source blob and the destination blob.

Changing a blob's tier from Hot to Cool or Archive is instantaneous, as is changing from Cool to Hot. Rehydrating a blob from the Archive tier to either the Hot or Cool tier can take up to 15 hours.

Keep in mind the following points when changing a blob's tier:

- You cannot call [Set Blob Tier](#) on a blob that uses an encryption scope. For more information about encryption scopes, see [Encryption scopes for Blob storage](#).
- If a blob's tier is inferred as Cool based on the storage account's default access tier and the blob is moved to the Archive tier, there's no early deletion charge.

- If a blob is explicitly moved to the Cool tier and then moved to the Archive tier, the early deletion charge applies.

The following table summarizes the approaches you can take to move blobs between various tiers.

ORIGIN/DESTINATION	HOT TIER	COOL TIER	ARCHIVE TIER
Hot tier	N/A	Change a blob's tier from Hot to Cool with <a href="#">Set Blob Tier</a> or <a href="#">Copy Blob</a> . <a href="#">Learn more...</a>  Move blobs to the Cool tier with a lifecycle management policy. <a href="#">Learn more...</a>	Change a blob's tier from Hot to Archive with <a href="#">Set Blob Tier</a> or <a href="#">Copy Blob</a> . <a href="#">Learn more...</a>  Archive blobs with a lifecycle management policy. <a href="#">Learn more...</a>
Cool tier	Change a blob's tier from Cool to Hot with <a href="#">Set Blob Tier</a> or <a href="#">Copy Blob</a> . <a href="#">Learn more...</a>  Move blobs to the Hot tier with a lifecycle management policy. <a href="#">Learn more...</a>	N/A	Change a blob's tier from Cool to Archive with <a href="#">Set Blob Tier</a> or <a href="#">Copy Blob</a> . <a href="#">Learn more...</a>  Archive blobs with a lifecycle management policy. <a href="#">Learn more...</a>
Archive tier	Rehydrate to Hot tier with <a href="#">Set Blob Tier</a> or <a href="#">Copy Blob</a> . <a href="#">Learn more...</a>	Rehydrate to Cool tier with <a href="#">Set Blob Tier</a> or <a href="#">Copy Blob</a> . <a href="#">Learn more...</a>	N/A

## Blob lifecycle management

Blob storage lifecycle management offers a rule-based policy that you can use to transition your data to the desired access tier when your specified conditions are met. You can also use lifecycle management to expire data at the end of its life. See [Optimize costs by automating Azure Blob Storage access tiers](#) to learn more.

### NOTE

Data stored in a premium block blob storage account cannot be tiered to Hot, Cool, or Archive using [Set Blob Tier](#) or using Azure Blob Storage lifecycle management. To move data, you must synchronously copy blobs from the block blob storage account to the Hot tier in a different account using the [Put Block From URL API](#) or a version of AzCopy that supports this API. The [Put Block From URL API](#) synchronously copies data on the server, meaning the call completes only once all the data is moved from the original server location to the destination location.

## Summary of access tier options

The following table summarizes the features of the Hot, Cool, and Archive access tiers.

	HOT TIER	COOL TIER	ARCHIVE TIER
Availability	99.9%	99%	Offline
Availability (RA-GRS reads)	99.99%	99.9%	Offline

	HOT TIER	COOL TIER	ARCHIVE TIER
<b>Usage charges</b>	Higher storage costs, but lower access and transaction costs	Lower storage costs, but higher access and transaction costs	Lowest storage costs, but highest access, and transaction costs
<b>Minimum recommended data retention period</b>	N/A	30 days <sup>1</sup>	180 days
<b>Latency (Time to first byte)</b>	Milliseconds	Milliseconds	Hours <sup>2</sup>
<b>Supported redundancy configurations</b>	All	All	LRS, GRS, and RA-GRS <sup>3</sup> only

<sup>1</sup> Objects in the Cool tier on general-purpose v2 accounts have a minimum retention duration of 30 days. For Blob Storage accounts, there's no minimum retention duration for the Cool tier.

<sup>2</sup> When rehydrating a blob from the Archive tier, you can choose either a standard or high rehydration priority option. Each offers different retrieval latencies and costs. For more information, see [Overview of blob rehydration from the Archive tier](#).

<sup>3</sup> For more information about redundancy configurations in Azure Storage, see [Azure Storage redundancy](#).

## Pricing and billing

All storage accounts use a pricing model for block blob storage that is based on a blob's tier. Keep in mind the billing considerations described in the following sections.

For more information about pricing for block blobs, see [Block blob pricing](#).

### Storage capacity costs

In addition to the amount of data stored, the cost of storing data varies depending on the access tier. The per-gigabyte capacity 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

A per-transaction charge applies to all tiers and 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. For more information on outbound data transfer charges, see [Bandwidth Pricing Details](#) page.

### Changing the default account access tier

Changing the account access tier results in tier change charges for all blobs that don't already have a tier explicitly set. For more information, see the following section, [Changing a blob's access tier](#).

### Changing a blob's access tier

Keep in mind the following billing impacts when changing a blob's tier:

- When a blob is uploaded or moved between tiers, it's charged at the corresponding rate immediately upon upload or tier change.
- When a blob is moved to a cooler tier (Hot to Cool, Hot to Archive, or Cool to Archive), the operation is billed as a write operation to the destination tier, where the write operation (per 10,000) and data write (per GB) charges of the destination tier apply.
- When a blob is moved to a warmer tier (Archive to Cool, Archive to Hot, or Cool to Hot), the operation is billed as a read from the source tier, where the read operation (per 10,000) and data retrieval (per GB) charges of the source tier apply. Early deletion charges for any blob moved out of the Cool or Archive tier may apply as well.
- While a blob is being rehydrated from the Archive tier, that blob's data is billed as archived data until the data is restored and the blob's tier changes to Hot or Cool.

The following table summarizes how tier changes are billed.

	WRITE CHARGES (OPERATION + ACCESS)	READ CHARGES (OPERATION + ACCESS)
<b>Set Blob Tier</b> operation	Hot to Cool Hot to Archive Cool to Archive	Archive to Cool Archive to Hot Cool to Hot

Changing the access tier for a blob when versioning is enabled, or if the blob has snapshots, may result in additional charges. For information about blobs with versioning enabled, see [Pricing and billing](#) in the blob versioning documentation. For information about blobs with snapshots, see [Pricing and billing](#) in the blob snapshots documentation.

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## Next steps

- [Set a blob's access tier](#)
- [Archive a blob](#)
- [Optimize costs by automatically managing the data lifecycle](#)

# Blob rehydration from the Archive tier

8/22/2022 • 10 minutes to read • [Edit Online](#)

While a blob is in the Archive access tier, it's considered to be offline, and can't be read or modified. In order 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. Microsoft recommends this option for most scenarios.
- [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 a large number of small blobs may 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.

To learn how to rehydrate an archived blob to an online tier, see [Rehydrate an archived blob to an online tier](#).

## 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 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 less than one hour for objects under 10 GB in size.

To check the rehydration priority while the rehydration operation is underway, call [Get Blob Properties](#) to return the value of the `x-ms-rehydrate-priority` header. The rehydration priority property returns either *Standard* or *High*.

Standard priority is the default rehydration option. A high-priority rehydration is faster, but also costs more than a standard-priority rehydration. A high-priority rehydration may take longer than one hour, depending on blob size and current demand. Microsoft recommends reserving high-priority rehydration for use in emergency data restoration situations.

While a standard-priority rehydration operation is pending, you can update the rehydration priority setting for a blob to *High* to rehydrate that blob more quickly. For example, if you're rehydrating a large number of blobs in bulk, you can specify *Standard* priority for all blobs for the initial operation, then increase the priority to *High* for any individual blobs that need to be brought online more quickly, up to the limit of 10 GiB per hour.

The rehydration priority setting can't be lowered from *High* to *Standard* for a pending operation. Keep in mind that updating the rehydration priority setting may have a billing impact.

To learn how to set and update the rehydration priority setting, see [Rehydrate an archived blob to an online tier](#).

For more information on pricing differences between standard-priority and high-priority rehydration requests, see [Pricing for Azure Blob Storage](#).

## Copy an archived blob to an online tier

The first option for moving a blob from the Archive tier to an online tier is to copy the archived blob to a new destination blob that is in either the Hot or Cool tier. You can use the [Copy Blob](#) operation to copy the blob. When you copy an archived blob to a new blob in 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 can't overwrite the source blob by copying to the same blob.

Microsoft recommends performing a copy operation in most scenarios where you need to move a blob from the Archive tier to an online tier, for the following reasons:

- A copy operation avoids the early deletion fee that is assessed if you change the tier of a blob from the Archive tier before the required 180-day period elapses. For more information, see [Archive access tier](#).
- If there's 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. A copy operation leaves the source blob in the Archive tier and creates a new blob with a different name and a new last modified time, so there's no risk that the rehydrated blob will be moved back to the Archive tier by the lifecycle policy.

Copying a blob from the Archive tier can take hours to complete depending on the rehydration priority selected. Behind the scenes, a blob copy operation reads your archived source blob to create a new online blob in the selected destination tier. The new blob may be visible when you list the blobs in the parent container before the rehydration operation is complete, but its tier will be set to Archive. The data isn't available until the read operation from the source blob in the Archive tier is complete and the blob's contents have been written to the new destination blob in an online tier. The new blob is an independent copy, so modifying or deleting it doesn't affect the source blob in the Archive tier.

To learn how to rehydrate a blob by copying it to an online tier, see [Rehydrate a blob with a copy operation](#).

#### IMPORTANT

Do not delete the source blob until the rehydration has completed successfully. If the source blob is deleted, then the destination blob may not finish copying. You can handle the event that is raised when the copy operation completes to know when it is safe to delete the source blob. For more information, see [Handle an event on blob rehydration](#).

Rehydrating an archived blob by copying it to an online destination tier is supported within the same storage account only for service versions prior to 2021-02-12. Beginning with service version 2021-02-12, you can rehydrate an archived blob by copying it to a different storage account, as long as the destination account is in the same region as the source account. Rehydration across storage accounts enables you to segregate your production data from your backup data, by maintaining them in separate accounts. Isolating archived data in a separate account can also help to mitigate costs from unintentional rehydration.

The target blob for the copy operation must be in an online tier (Hot or Cool). You can't 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 SOURCE	COOL TIER SOURCE	ARCHIVE TIER SOURCE
Hot tier destination	Supported	Supported	Supported across accounts in the same region with version 2021-02-12 and later. Supported within the same storage account only for earlier versions. Requires blob rehydration.
Cool tier destination	Supported	Supported	Supported across accounts in the same region with version 2021-02-12 and later. Supported within the same storage account only for earlier versions. Requires blob rehydration.
Archive tier destination	Supported	Supported	Not supported

### Rehydrate from a secondary region

If you've configured your storage account to use read-access geo-redundant storage (RA-GRS), then you can use the [Copy Blob](#) operation to rehydrate blobs in the secondary region to another storage account that is located in that same secondary region. See [Rehydrate from a secondary region](#).

To learn more about obtaining read access to secondary regions, see [Read access to data in the secondary region](#).

## 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 can't be canceled. During the rehydration operation, the blob's access tier setting continues to show as archived until the rehydration process is complete. When the rehydration operation is complete, the blob's access tier property updates to reflect the new tier.

To learn how to rehydrate a blob by changing its tier to an online tier, see [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.

To avoid this scenario, rehydrate the archived blob by copying it instead, as described in the [Copy an archived blob to an online tier](#) section. Performing a copy operation creates a new instance of the blob with an updated last modified time, so it won't trigger the lifecycle management policy.

## Check the status of a blob rehydration operation

During the blob rehydration operation, you can call the [Get Blob Properties](#) operation to check its status. To learn how to check the status of a rehydration operation, see [Check the status of a rehydration operation](#).

## Handle an event on blob rehydration

Rehydration of an archived blob may take up to 15 hours, and it is inefficient to repeatedly poll [Get Blob](#)

**Properties** to determine whether rehydration is complete. Microsoft recommends that you use [Azure Event Grid](#) to capture the event that fires when rehydration is complete for better performance and cost optimization.

Azure Event Grid raises one of the following two events on blob rehydration, depending on which operation was used to rehydrate the blob:

- The **Microsoft.Storage.BlobCreated** event fires when a blob is created. In the context of blob rehydration, this event fires when a [Copy Blob](#) operation creates a new destination blob in either the Hot or Cool tier and the blob's data is fully rehydrated from the Archive tier.
- The **Microsoft.Storage.BlobTierChanged** event fires when a blob's tier is changed. In the context of blob rehydration, this event fires when a [Set Blob Tier](#) operation successfully changes an archived blob's tier to the Hot or Cool tier.

To learn how to capture an event on rehydration and send it to an Azure Function event handler, see [Run an Azure Function in response to a blob rehydration event](#).

For more information on handling events in Blob Storage, see [Reacting to Azure Blob storage events](#) and [Azure Blob Storage as Event Grid source](#).

## Pricing and billing

A rehydration operation with [Set Blob Tier](#) is billed for data read transactions and data retrieval size. A high-priority rehydration has higher operation and data retrieval costs compared to standard priority. High-priority rehydration shows up as a separate line item on your bill. If a high-priority request to return an archived blob of a few gigabytes takes more than five hours, you won't be charged the high-priority retrieval rate. However, standard retrieval rates still apply.

Copying an archived blob to an online tier with [Copy Blob](#) is billed for data read transactions and data retrieval size. Creating the destination blob in an online tier is billed for data write transactions. Early deletion fees don't apply when you copy to an online blob because the source blob remains unmodified in the Archive tier. High-priority retrieval charges do apply if selected.

Blobs in the Archive tier should be stored for a minimum of 180 days. Deleting or changing the tier of an archived blob before the 180-day period elapses incurs an early deletion fee. For example, if a blob is moved to the Archive tier and then deleted or moved to the Hot tier after 45 days, you'll be charged an early deletion fee equivalent to 135 (180 minus 45) days of storing that blob in the Archive tier. For more information, see [Archive access tier](#).

For more information about pricing for block blobs and data rehydration, see [Azure Storage Pricing](#). For more information on outbound data transfer charges, see [Data Transfers Pricing Details](#).

## See also

- [Hot, Cool, and Archive access tiers for blob data](#)
- [Archive a blob](#)
- [Rehydrate an archived blob to an online tier](#)
- [Run an Azure Function in response to a blob rehydration event](#)
- [Reacting to Blob storage events](#)

# Optimize costs by automatically managing the data lifecycle

8/22/2022 • 14 minutes to read • [Edit Online](#)

Data sets have unique lifecycles. Early in the lifecycle, people access some data often. But the need for access often drops drastically as the data ages. Some data remains idle in the cloud and is rarely accessed once stored. Some data sets expire days or months after creation, while other data sets are actively read and modified throughout their lifetimes. Azure Storage lifecycle management offers a rule-based policy that you can use to transition blob data to the appropriate access tiers or to expire data at the end of the data lifecycle.

With the lifecycle management policy, you can:

- Transition blobs from cool to hot immediately when they're accessed, to optimize for performance.
- Transition current versions of a blob, previous versions of a blob, or blob snapshots to a cooler storage tier if these objects haven't been accessed or modified for a period of time, to optimize for cost. In this scenario, the lifecycle management policy can move objects from hot to cool, from hot to archive, or from cool to archive.
- Delete current versions of a blob, previous versions of a blob, or blob snapshots at the end of their lifecycles.
- Define rules to be run once per day at the storage account level.
- Apply rules to containers or to a subset of blobs, using name prefixes or [blob index tags](#) as filters.

Consider a scenario where data is frequently accessed during the early stages of the lifecycle, but only occasionally after two weeks. Beyond the first month, the data set is rarely accessed. In this scenario, hot storage is best during the early stages. Cool storage is most appropriate for occasional access. Archive storage is the best tier option after the data ages over a month. By moving data to the appropriate storage tier based on its age with lifecycle management policy rules, you can design the least expensive solution for your needs.

Lifecycle management policies are supported for block blobs and append blobs in general-purpose v2, premium block blob, and Blob Storage accounts. Lifecycle management doesn't affect system containers such as the `$logs` or `$web` containers.

## IMPORTANT

If a data set needs to be readable, do not set a policy to move blobs to the archive tier. Blobs in the archive tier cannot be read unless they are first rehydrated, a process which may be time-consuming and expensive. For more information, see [Overview of blob rehydration from the archive tier](#).

## Lifecycle management policy definition

A lifecycle management policy is a collection of rules in a JSON document. The following sample JSON shows a complete rule definition:

```
{
 "rules": [
 {
 "name": "rule1",
 "enabled": true,
 "type": "Lifecycle",
 "definition": {...}
 },
 {
 "name": "rule2",
 "type": "Lifecycle",
 "definition": {...}
 }
]
}
```

A policy is a collection of rules, as described in the following table:

PARAMETER NAME	PARAMETER TYPE	NOTES
<code>rules</code>	An array of rule objects	At least one rule is required in a policy. You can define up to 100 rules in a policy.

Each rule within the policy has several parameters, described in the following table:

PARAMETER NAME	PARAMETER TYPE	NOTES	REQUIRED
<code>name</code>	String	A rule name can include up to 256 alphanumeric characters. Rule name is case-sensitive. It must be unique within a policy.	True
<code>enabled</code>	Boolean	An optional boolean to allow a rule to be temporarily disabled. Default value is true if it's not set.	False
<code>type</code>	An enum value	The current valid type is <code>Lifecycle</code> .	True
<code>definition</code>	An object that defines the lifecycle rule	Each definition is made up of a filter set and an action set.	True

## Lifecycle management rule definition

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.

### Sample rule

The following sample rule filters the account to run the actions on objects that exist inside `sample-container` and start with `blob1`.

- Tier blob to cool tier 30 days after last modification

- Tier blob to archive tier 90 days after last modification
- Delete blob 2,555 days (seven years) after last modification
- Delete previous versions 90 days after creation

```
{
 "rules": [
 {
 "enabled": true,
 "name": "sample-rule",
 "type": "Lifecycle",
 "definition": {
 "actions": {
 "version": {
 "delete": {
 "daysAfterCreationGreaterThanOrEqual": 90
 }
 },
 "baseBlob": {
 "tierToCool": {
 "daysAfterModificationGreaterThanOrEqual": 30
 },
 "tierToArchive": {
 "daysAfterModificationGreaterThanOrEqual": 90
 },
 "delete": {
 "daysAfterModificationGreaterThanOrEqual": 2555
 }
 }
 },
 "filters": {
 "blobTypes": [
 "blockBlob"
],
 "prefixMatch": [
 "sample-container/blob1"
]
 }
 }
 }
]
}
```

#### NOTE

The **baseBlob** element in a lifecycle management policy refers to the current version of a blob. The **version** element refers to a previous version.

#### 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	NOTES	IS REQUIRED
blobTypes	An array of predefined enum values.	The current release supports <b>blockBlob</b> and <b>appendBlob</b> . Only delete is supported for <b>appendBlob</b> , set tier isn't supported.	Yes

Filter Name	Filter Type	Notes	Is Required
prefixMatch	An array of strings for prefixes to be matched. Each rule can define up to 10 case-sensitive prefixes. A prefix string must start with a container name. For example, if you want to match all blobs under <code>https://myaccount.blob.core.windows.net/sample-container/blob1/...</code> for a rule, the prefixMatch is <code>sample-container/blob1</code> .	If you don't define prefixMatch, the rule applies to all blobs within the storage account.	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. For example, if you want to match all blobs with <code>Project = Contoso</code> under <code>https://myaccount.blob.core.windows.net/</code> for a rule, the blobIndexMatch is <code>{"name": "Project", "op": "==", "value": "Contoso"}</code> .	If you don't define blobIndexMatch, the rule applies to all blobs within the storage account.	No

To learn more about the blob index feature together with known issues and limitations, see [Manage and find data on Azure Blob Storage with blob index](#).

## Rule actions

Actions are applied to the filtered blobs when the run condition is met.

Lifecycle management supports tiering and deletion of current versions, previous versions, and blob snapshots. Define at least one action for each rule.

Action	Current Version	Snapshot	Previous Versions
tierToCool	Supported for <code>blockBlob</code>	Supported	Supported
enableAutoTierToHotFromCool	Supported for <code>blockBlob</code>	Not supported	Not supported
tierToArchive	Supported for <code>blockBlob</code>	Supported	Supported
delete <sup>1</sup>	Supported for <code>blockBlob</code> and <code>appendBlob</code>	Supported	Supported

<sup>1</sup> When applied to an account with a hierarchical namespace enabled, a delete action removes empty directories. If the directory isn't empty, then the delete action removes objects that meet the policy conditions within the first 24-hour cycle. If that action results in an empty directory that also meets the policy conditions, then that directory will be removed within the next 24-hour cycle, and so on.

#### NOTE

If you define more than one action on the same blob, lifecycle management applies the least expensive action to the blob.

For example, action `delete` is cheaper than action `tierToArchive`. Action `tierToArchive` is cheaper than action `tierToCool`.

The run conditions are based on age. Current versions use the last modified time or last access time, previous versions use the version creation time, and blob snapshots use the snapshot creation time to track age.

ACTION RUN CONDITION	CONDITION VALUE	DESCRIPTION
<code>daysAfterModificationGreaterThan</code>	Integer value indicating the age in days	The condition for actions on a current version of a blob
<code>daysAfterCreationGreaterThan</code>	Integer value indicating the age in days	The condition for actions on a previous version of a blob or a blob snapshot
<code>daysAfterLastAccessTimeGreaterThan</code>	Integer value indicating the age in days	The condition for a current version of a blob when access tracking is enabled

## Examples of lifecycle policies

The following examples demonstrate how to address common scenarios with lifecycle policy rules.

### Move aging data to a cooler tier

This example shows how to transition block blobs prefixed with `sample-container/blob1` or `container2/blob2`.

The policy transitions blobs that haven't been modified in over 30 days to cool storage, and blobs not modified in 90 days to the archive tier:

```
{
 "rules": [
 {
 "name": "agingRule",
 "enabled": true,
 "type": "Lifecycle",
 "definition": {
 "filters": {
 "blobTypes": ["blockBlob"],
 "prefixMatch": ["sample-container/blob1", "container2/blob2"]
 },
 "actions": {
 "baseBlob": {
 "tierToCool": { "daysAfterModificationGreaterThan": 30 },
 "tierToArchive": { "daysAfterModificationGreaterThan": 90 }
 }
 }
 }
 }
]
}
```

### Move data based on last accessed time

You can enable last access time tracking to keep a record of when your blob is last read or written and as a filter to manage tiering and retention of your blob data. To learn how to enable last access time tracking, see [Optionally enable access time tracking](#).

When last access time tracking is enabled, the blob property called `LastAccessTime` is updated when a blob is

read or written. A [Get Blob](#) operation is considered an access operation. [Get Blob Properties](#), [Get Blob Metadata](#), and [Get Blob Tags](#) aren't access operations, and therefore don't update the last access time.

To minimize the effect on read access latency, only the first read of the last 24 hours updates the last access time. Subsequent reads in the same 24-hour period don't update the last access time. If a blob is modified between reads, the last access time is the more recent of the two values.

In the following example, blobs are moved to cool storage if they haven't been accessed for 30 days. The `enableAutoTierToHotFromCool` property is a Boolean value that indicates whether a blob should automatically be tiered from cool back to hot if it's accessed again after being tiered to cool.

```
{
 "enabled": true,
 "name": "last-accessed-thirty-days-ago",
 "type": "Lifecycle",
 "definition": {
 "actions": {
 "baseBlob": {
 "enableAutoTierToHotFromCool": true,
 "tierToCool": {
 "daysAfterLastAccessTimeGreater Than": 30
 }
 }
 },
 "filters": {
 "blobTypes": [
 "blockBlob"
],
 "prefixMatch": [
 "mylifecyclecontainer/log"
]
 }
 }
}
```

## Archive data after ingest

Some data stays idle in the cloud and is rarely, if ever, accessed. The following lifecycle policy is configured to archive data shortly after it's ingested. This example transitions block blobs in a container named `archivecontainer` into an archive tier. The transition is accomplished by acting on blobs 0 days after last modified time:

```
{
 "rules": [
 {
 "name": "archiveRule",
 "enabled": true,
 "type": "Lifecycle",
 "definition": {
 "filters": {
 "blobTypes": ["blockBlob"],
 "prefixMatch": ["archivecontainer"]
 },
 "actions": {
 "baseBlob": {
 "tierToArchive": { "daysAfterModificationGreaterThan": 0 }
 }
 }
 }
 }
]
}
```

#### **NOTE**

Microsoft recommends that you upload your blobs directly to the archive tier for greater efficiency. You can specify the archive tier in the *x-ms-access-tier* header on the [Put Blob](#) or [Put Block List](#) operation. The *x-ms-access-tier* header is supported with REST version 2018-11-09 and newer or the latest blob storage client libraries.

#### **Expire data based on age**

Some data is expected to expire days or months after creation. You can configure a lifecycle management policy to expire data by deletion based on data age. The following example shows a policy that deletes all block blobs that haven't been modified in the last 365 days.

```
{
 "rules": [
 {
 "name": "expirationRule",
 "enabled": true,
 "type": "Lifecycle",
 "definition": {
 "filters": {
 "blobTypes": ["blockBlob"]
 },
 "actions": {
 "baseBlob": {
 "delete": { "daysAfterModificationGreaterThan": 365 }
 }
 }
 }
 }
]
}
```

#### **Delete data with blob index tags**

Some data should only be expired if explicitly marked for deletion. You can configure a lifecycle management policy to expire data that are tagged with blob index key/value attributes. The following example shows a policy that deletes all block blobs tagged with `Project = Contoso`. To learn more about blob index, see [Manage and find data on Azure Blob Storage with blob index](#).

```
{
 "rules": [
 {
 "enabled": true,
 "name": "DeleteContosoData",
 "type": "Lifecycle",
 "definition": {
 "actions": {
 "baseBlob": {
 "delete": {
 "daysAfterModificationGreaterThanOrEqual": 0
 }
 }
 },
 "filters": {
 "blobIndexMatch": [
 {
 "name": "Project",
 "op": "==",
 "value": "Contoso"
 }
],
 "blobTypes": [
 "blockBlob"
]
 }
 }
 }
]
}
```

## Manage previous versions

For data that is modified and accessed regularly throughout its lifetime, you can enable blob storage versioning to automatically maintain previous versions of an object. You can create a policy to tier or delete previous versions. The version age is determined by evaluating the version creation time. This policy rule moves previous versions within container `activedata` that are 90 days or older after version creation to the cool tier, and deletes previous versions that are 365 days or older.

```
{
 "rules": [
 {
 "enabled": true,
 "name": "versionrule",
 "type": "Lifecycle",
 "definition": {
 "actions": {
 "version": {
 "tierToCool": {
 "daysAfterCreationGreaterThanOrEqual": 90
 },
 "delete": {
 "daysAfterCreationGreaterThanOrEqual": 365
 }
 }
 },
 "filters": {
 "blobTypes": [
 "blockBlob"
],
 "prefixMatch": [
 "activedata"
]
 }
 }
 }
]
}
```

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## Regional availability and pricing

The lifecycle management feature is available in all Azure regions.

Lifecycle management policies are free of charge. Customers are billed for standard operation costs for the [Set Blob Tier](#) API calls. Delete operations are free. However, other Azure services and utilities such as [Microsoft Defender for Storage](#) may charge for operations that are managed through a lifecycle policy.

Each update to a blob's last access time is billed under the [other operations](#) category.

For more information about pricing, see [Block Blob pricing](#).

## FAQ

### I created a new policy. Why do the actions not run immediately?

The platform runs the lifecycle policy once a day. Once you configure a policy, it can take up to 24 hours for some actions to run for the first time.

### If I update an existing policy, how long does it take for the actions to run?

The updated policy takes up to 24 hours to go into effect. Once the policy is in effect, it could take up to 24 hours for the actions to run. Therefore, the policy actions may take up to 48 hours to complete. If the update is to disable or delete a rule, and enableAutoTierToHotFromCool was used, auto-tiering to Hot tier will still happen.

For example, set a rule including enableAutoTierToHotFromCool based on last access. If the rule is disabled/deleted, and a blob is currently in cool and then accessed, it will move back to Hot as that is applied on access outside of lifecycle management. The blob won't then move from Hot to Cool given the lifecycle management rule is disabled/deleted. The only way to prevent autoTierToHotFromCool is to turn off last access time tracking.

### I manually rehydrated an archived blob. How do I prevent it from being moved back to the Archive tier temporarily?

When a blob is moved from one access tier to another, its last modification time doesn't change. If you manually rehydrate an archived blob to hot tier, it would be moved back to archive tier by the lifecycle management engine. Disable the rule that affects this blob temporarily to prevent it from being archived again. Re-enable the rule when the blob can be safely moved back to archive tier. You may also copy the blob to another location if it needs to stay in hot or cool tier permanently.

### The blob prefix match string didn't apply the policy to the expected blobs

The blob prefix match field of a policy is a full or partial blob path, which is used to match the blobs you want the policy actions to apply to. The path must start with the container name. If no prefix match is specified, then the policy will apply to all the blobs in the storage account. The format of the prefix match string is

[container name]/[blob name].

Keep in mind the following points about the prefix match string:

- A prefix match string like *container1/* applies to all blobs in the container named *container1*. A prefix match string of *container1*, without the trailing forward slash character (/), applies to all blobs in all containers where the container name begins with the string *container1*. The prefix will match containers named *container11*, *container1234*, *container1ab*, and so on.
- A prefix match string of *container1/sub1/* applies to all blobs in the container named *container1* that begin with the string *sub1/*. For example, the prefix will match blobs named *container1/sub1/test.txt* or *container1/sub1/sub2/test.txt*.
- The asterisk character `*` is a valid character in a blob name. If the asterisk character is used in a prefix, then the prefix will match blobs with an asterisk in their names. The asterisk doesn't function as a wildcard character.
- The question mark character `?` is a valid character in a blob name. If the question mark character is used in a prefix, then the prefix will match blobs with a question mark in their names. The question mark doesn't function as a wildcard character.
- The prefix match considers only positive (=) logical comparisons. Negative (!=) logical comparisons are ignored.

### Is there a way to identify the time at which the policy will be executing?

Unfortunately, there's no way to track the time at which the policy will be executing, as it's a background scheduling process. However, the platform will run the policy once per day.

## Next steps

- [Configure a lifecycle management policy](#)
- [Hot, Cool, and Archive access tiers for blob data](#)
- [Manage and find data on Azure Blob Storage with blob index](#)

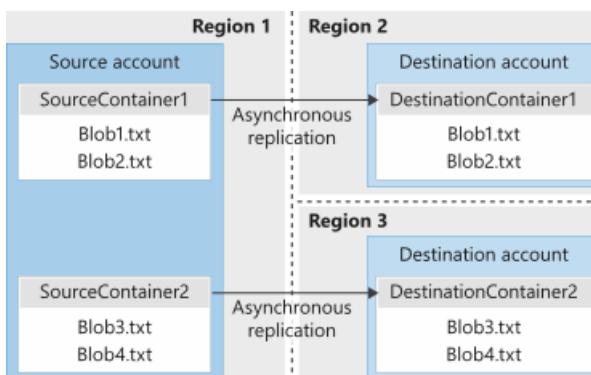
# Object replication for block blobs

8/22/2022 • 14 minutes to read • [Edit Online](#)

Object replication asynchronously copies block blobs between a source storage account and a destination account. Some scenarios supported by object replication include:

- **Minimizing latency.** Object replication can reduce latency for read requests by enabling clients to consume data from a region that is in closer physical proximity.
- **Increase efficiency for compute workloads.** With object replication, compute workloads can process the same sets of block blobs in different regions.
- **Optimizing data distribution.** You can process or analyze data in a single location and then replicate just the results to additional regions.
- **Optimizing costs.** After your data has been replicated, you can reduce costs by moving it to the archive tier using life cycle management policies.

The following diagram shows how object replication replicates block blobs from a source storage account in one region to destination accounts in two different regions.



To learn how to configure object replication, see [Configure object replication](#).

## Prerequisites and caveats for object replication

Object replication requires that the following Azure Storage features are also enabled:

- **Change feed:** Must be enabled on the source account. To learn how to enable change feed, see [Enable and disable the change feed](#).
- **Blob versioning:** Must be enabled on both the source and destination accounts. To learn how to enable versioning, see [Enable and manage blob versioning](#).

Enabling change feed and blob versioning may incur additional costs. For more information, see the [Azure Storage pricing page](#).

Object replication is supported for general-purpose v2 storage accounts and premium block blob accounts. Both the source and destination accounts must be either general-purpose v2 or premium block blob accounts. Object replication supports block blobs only; append blobs and page blobs aren't supported.

Object replication is supported for accounts that are encrypted with customer-managed keys. For more information about customer-managed keys, see [Customer-managed keys for Azure Storage encryption](#).

Object replication isn't supported for blobs in the source account that are encrypted with a customer-provided key. For more information about customer-provided keys, see [Provide an encryption key on a request to Blob](#)

storage.

Customer-managed failover isn't supported for either the source or the destination account in an object replication policy.

## How object replication works

Object replication asynchronously copies block blobs in a container according to rules that you configure. The contents of the blob, any versions associated with the blob, and the blob's metadata and properties are all copied from the source container to the destination container.

### IMPORTANT

Because block blob data is replicated asynchronously, the source account and destination account are not immediately in sync. There's currently no SLA on how long it takes to replicate data to the destination account. You can check the replication status on the source blob to determine whether replication is complete. For more information, see [Check the replication status of a blob](#).

### Blob versioning

Object replication requires that blob versioning is enabled on both the source and destination accounts. When a replicated blob in the source account is modified, a new version of the blob is created in the source account that reflects the previous state of the blob, before modification. The current version in the source account reflects the most recent updates. Both the current version and any previous versions are replicated to the destination account. For more information about how write operations affect blob versions, see [Versioning on write operations](#).

When a blob in the source account is deleted, the current version of the blob becomes a previous version, and there's no longer a current version. All existing previous versions of the blob are preserved. This state is replicated to the destination account. For more information about how to delete operations affect blob versions, see [Versioning on delete operations](#).

### Snapshots

Object replication doesn't support blob snapshots. Any snapshots on a blob in the source account aren't replicated to the destination account.

## Blob index tags

Object replication does not copy the source blob's index tags to the destination blob.

### Blob tiering

Object replication is supported when the source and destination accounts are in the hot or cool tier. The source and destination accounts may be in different tiers. However, object replication will fail if a blob in either the source or destination account has been moved to the archive tier. For more information on blob tiers, see [Hot, Cool, and Archive access tiers for blob data](#).

### Immutable blobs

Immutability policies for Azure Blob Storage include time-based retention policies and legal holds. When an immutability policy is in effect on the destination account, object replication may be affected. For more information about immutability policies, see [Store business-critical blob data with immutable storage](#).

If a container-level immutability policy is in effect for a container in the destination account, and an object in the source container is updated or deleted, then the operation on the source container may succeed, but replication of that operation to the destination container will fail. For more information about which operations are prohibited with an immutability policy that is scoped to a container, see [Scenarios with container-level scope](#).

If a version-level immutability policy is in effect for a blob version in the destination account, and a delete or update operation is performed on the blob version in the source container, then the operation on the source object may succeed, but replication of that operation to the destination object will fail. For more information about which operations are prohibited with an immutability policy that is scoped to a container, see [Scenarios with version-level scope](#).

## Object replication policies and rules

When you configure object replication, you create a replication policy that specifies the source storage account and the destination account. A replication policy includes one or more rules that specify a source container and a destination container and indicate which block blobs in the source container will be replicated.

After you configure object replication, Azure Storage checks the change feed for the source account periodically and asynchronously replicates any write or delete operations to the destination account. Replication latency depends on the size of the block blob being replicated.

### Replication policies

When you configure object replication, you create a replication policy on the destination account via the Azure Storage resource provider. After the replication policy is created, Azure Storage assigns it a policy ID. You must then associate that replication policy with the source account by using the policy ID. The policy ID on the source and destination accounts must be the same in order for replication to take place.

A source account can replicate to no more than two destination accounts, with one policy for each destination account. Similarly, an account may serve as the destination account for no more than two replication policies.

The source and destination accounts may be in the same region or in different regions. They may also reside in the same subscription or in different subscriptions. Optionally, the source and destination accounts may reside in different Azure Active Directory (Azure AD) tenants. Only one replication policy may be created for each source account/destination account pair.

### Replication rules

Replication rules specify how Azure Storage will replicate blobs from a source container to a destination container. You can specify up to 1000 replication rules for each replication policy. Each replication rule defines a single source and destination container, and each source and destination container can be used in only one rule, meaning that a maximum of 1000 source containers and 1000 destination containers may participate in a single replication policy.

When you create a replication rule, by default only new block blobs that are subsequently added to the source container are copied. You can specify that both new and existing block blobs are copied, or you can define a custom copy scope that copies block blobs created from a specified time onward.

You can also specify one or more filters as part of a replication rule to filter block blobs by prefix. When you specify a prefix, only blobs matching that prefix in the source container will be copied to the destination container.

The source and destination containers must both exist before you can specify them in a rule. After you create the replication policy, write operations to the destination container aren't permitted. Any attempts to write to the destination container fail with error code 409 (Conflict). To write to a destination container for which a replication rule is configured, you must either delete the rule that is configured for that container, or remove the replication policy. Read and delete operations to the destination container are permitted when the replication policy is active.

You can call the [Set Blob Tier](#) operation on a blob in the destination container to move it to the archive tier. For more information about the archive tier, see [Hot, Cool, and Archive access tiers for blob data](#).

# Policy definition file

An object replication policy is defined by JSON file. You can get the policy definition file from an existing object replication policy. You can also create an object replication policy by uploading a policy definition file.

## Sample policy definition file

The following example defines a replication policy on the destination account with a single rule that matches the prefix *b* and sets the minimum creation time for blobs that are to be replicated. Remember to replace values in angle brackets with your own values:

```
{
 "properties": {
 "policyId": "default",
 "sourceAccount": "/subscriptions/<subscriptionId>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>",
 "destinationAccount": "/subscriptions/<subscriptionId>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>",
 "rules": [
 {
 "ruleId": "",
 "sourceContainer": "<source-container>",
 "destinationContainer": "<destination-container>",
 "filters": {
 "prefixMatch": [
 "b"
],
 "minCreationTime": "2021-08-028T00:00:00Z"
 }
 }
]
 }
}
```

## Specify full resource IDs for the source and destination accounts

When you create the policy definition file, specify the full Azure Resource Manager resource IDs for the **sourceAccount** and **destinationAccount** entries, as shown in the example in the previous section. To learn how to locate the resource ID for a storage account, see [Get the resource ID for a storage account](#).

The full resource ID is in the following format:

```
/subscriptions/<subscriptionId>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>
```

The policy definition file previously required only the account name, instead of the full resource ID for the storage account. With the introduction of the **AllowCrossTenantReplication** security property in version 2021-02-01 of the Azure Storage resource provider REST API, you must now provide the full resource ID for any object replication policies that are created when cross-tenant replication is disallowed for a storage account that participates in the replication policy. Azure Storage uses the full resource ID to verify whether the source and destination accounts reside within the same tenant. To learn more about disallowing cross-tenant replication policies, see [Prevent replication across Azure AD tenants](#).

While providing only the account name is still supported when cross-tenant replication is allowed for a storage account, Microsoft recommends always providing the full resource ID as a best practice. All previous versions of the Azure Storage resource provider REST API support using the full resource ID path in object replication policies.

The following table describes what happens when you create a replication policy with the full resource ID specified, versus the account name, in the scenarios where cross-tenant replication is allowed or disallowed for

the storage account.

STORAGE ACCOUNT IDENTIFIER IN POLICY DEFINITION	CROSS-TENANT REPLICATION ALLOWED	CROSS-TENANT REPLICATION DISALLOWED
Full resource ID	Same-tenant policies can be created.  Cross-tenant policies can be created.	Same-tenant policies can be created.  Cross-tenant policies can't be created.
Account name only	Same-tenant policies can be created.  Cross-tenant policies can be created.	Neither same-tenant nor cross-tenant policies can be created. An error occurs, because Azure Storage can't verify that source and destination accounts are in the same tenant. The error indicates that you must specify the full resource ID for the <b>sourceAccount</b> and <b>destinationAccount</b> entries in the policy definition file.

## Specify the policy and rule IDs

The following table summarizes which values to use for the **policyId** and **ruleId** entries in the policy definition file in each scenario.

WHEN YOU'RE CREATING THE POLICY DEFINITION FILE FOR THIS ACCOUNT...	SET THE POLICY ID TO THIS VALUE	SET RULE IDS TO THIS VALUE
Destination account	The string value <i>default</i> . Azure Storage will create the policy ID value for you.	An empty string. Azure Storage will create the rule ID values for you.
Source account	The value of the policy ID returned when you download the policy definition file for the destination account.	The values of the rule IDs returned when you download the policy definition file for the destination account.

## Prevent replication across Azure AD tenants

An Azure Active Directory (Azure AD) tenant is a dedicated instance of Azure AD that represents an organization for identity and access management. Each Azure subscription has a trust relationship with a single Azure AD tenant. All resources in a subscription, including storage accounts, are associated with the same Azure AD tenant. For more information, see [What is Azure Active Directory?](#)

By default, a user with appropriate permissions can configure object replication with a source storage account that is in one Azure AD tenant and a destination account that is in a different tenant. If your security policies require that you restrict object replication to storage accounts that reside within the same tenant only, you can disallow replication across tenants by setting a security property, the **AllowCrossTenantReplication** property (preview). When you disallow cross-tenant object replication for a storage account, then for any object replication policy that is configured with that storage account as the source or destination account, Azure Storage requires that both the source and destination accounts reside within the same Azure AD tenant. For more information about disallowing cross-tenant object replication, see [Prevent object replication across Azure Active Directory tenants](#).

To disallow cross-tenant object replication for a storage account, set the **AllowCrossTenantReplication** property to *false*. If the storage account doesn't currently participate in any cross-tenant object replication policies, then setting the **AllowCrossTenantReplication** property to *false* prevents future configuration of cross-tenant object replication policies with this storage account as the source or destination.

If the storage account currently participates in one or more cross-tenant object replication policies, then setting the **AllowCrossTenantReplication** property to *false* isn't permitted. You must delete the existing cross-tenant policies before you can disallow cross-tenant replication.

By default, the **AllowCrossTenantReplication** property isn't set for a storage account, and its value is *null*, which is equivalent to *true*. When the value of the **AllowCrossTenantReplication** property for a storage account is *null* or *true*, then authorized users can configure cross-tenant object replication policies with this account as the source or destination. For more information about how to configure cross-tenant policies, see [Configure object replication for block blobs](#).

You can use Azure Policy to audit a set of storage accounts to ensure that the **AllowCrossTenantReplication** property is set to prevent cross-tenant object replication. You can also use Azure Policy to enforce governance for a set of storage accounts. For example, you can create a policy with the deny effect to prevent a user from creating a storage account where the **AllowCrossTenantReplication** property is set to *true*, or from modifying an existing storage account to change the property value to *true*.

## Replication status

You can check the replication status for a blob in the source account. For more information, see [Check the replication status of a blob](#).

If the replication status for a blob in the source account indicates failure, then investigate the following possible causes:

- Make sure that the object replication policy is configured on the destination account.
- Verify that the destination account still exists.
- Verify that the destination container still exists.
- Verify that the destination container is not in the process of being deleted, or has not just been deleted. Deleting a container may take up to 30 seconds.
- Verify that the destination container is still participating in the object replication policy.
- If the source blob has been encrypted with a customer-provided key as part of a write operation, then object replication will fail. For more information about customer-provided keys, see [Provide an encryption key on a request to Blob storage](#).
- Check whether the source or destination blob has been moved to the Archive tier. Archived blobs cannot be replicated via object replication. For more information about the Archive tier, see [Hot, Cool, and Archive access tiers for blob data](#).
- Verify that destination container or blob is not protected by an immutability policy. Keep in mind that a container or blob can inherit an immutability policy from its parent. For more information about immutability policies, see [Overview of immutable storage for blob data](#).

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## Billing

Object replication incurs additional costs on read and write transactions against the source and destination accounts, as well as egress charges for the replication of data from the source account to the destination account and read charges to process change feed.

## Next steps

- [Configure object replication](#)
- [Prevent object replication across Azure Active Directory tenants](#)
- [Blob versioning](#)
- [Change feed support in Azure Blob Storage](#)

# Performance and scalability checklist for Blob storage

8/22/2022 • 23 minutes to read • [Edit Online](#)

Microsoft has developed a number of proven practices for developing high-performance applications with Blob storage. This checklist identifies key practices that developers can follow to optimize performance. Keep these practices in mind while you are designing your application and throughout the process.

Azure Storage has scalability and performance targets for capacity, transaction rate, and bandwidth. For more information about Azure Storage scalability targets, see [Scalability and performance targets for standard storage accounts](#) and [Scalability and performance targets for Blob storage](#).

## Checklist

This article organizes proven practices for performance into a checklist you can follow while developing your Blob storage application.

DONE	CATEGORY	DESIGN CONSIDERATION
	Scalability targets	<a href="#">Can you design your application to use no more than the maximum number of storage accounts?</a>
	Scalability targets	<a href="#">Are you avoiding approaching capacity and transaction limits?</a>
	Scalability targets	<a href="#">Are a large number of clients accessing a single blob concurrently?</a>
	Scalability targets	<a href="#">Is your application staying within the scalability targets for a single blob?</a>
	Partitioning	<a href="#">Is your naming convention designed to enable better load-balancing?</a>
	Networking	<a href="#">Do client-side devices have sufficiently high bandwidth and low latency to achieve the performance needed?</a>
	Networking	<a href="#">Do client-side devices have a high quality network link?</a>
	Networking	<a href="#">Is the client application in the same region as the storage account?</a>
	Direct client access	<a href="#">Are you using shared access signatures (SAS) and cross-origin resource sharing (CORS) to enable direct access to Azure Storage?</a>

DONE	CATEGORY	DESIGN CONSIDERATION
	Caching	Is your application caching data that is frequently accessed and rarely changed?
	Caching	Is your application batching updates by caching them on the client and then uploading them in larger sets?
	.NET configuration	Are you using .NET Core 2.1 or later for optimum performance?
	.NET configuration	Have you configured your client to use a sufficient number of concurrent connections?
	.NET configuration	For .NET applications, have you configured .NET to use a sufficient number of threads?
	Parallelism	Have you ensured that parallelism is bounded appropriately so that you don't overload your client's capabilities or approach the scalability targets?
	Tools	Are you using the latest versions of Microsoft-provided client libraries and tools?
	Retries	Are you using a retry policy with an exponential backoff for throttling errors and timeouts?
	Retries	Is your application avoiding retries for non-retryable errors?
	Copying blobs	Are you copying blobs in the most efficient manner?
	Copying blobs	Are you using the latest version of AzCopy for bulk copy operations?
	Copying blobs	Are you using the Azure Data Box family for importing large volumes of data?
	Content distribution	Are you using a CDN for content distribution?
	Use metadata	Are you storing frequently used metadata about blobs in their metadata?
	Uploading quickly	When trying to upload one blob quickly, are you uploading blocks in parallel?

DONE	CATEGORY	DESIGN CONSIDERATION
	Uploading quickly	When trying to upload many blobs quickly, are you uploading blobs in parallel?
	Blob type	Are you using page blobs or block blobs when appropriate?

## Scalability targets

If your application approaches or exceeds any of the scalability targets, it may encounter increased transaction latencies or throttling. When Azure Storage throttles your application, the service begins to return 503 (Server busy) or 500 (Operation timeout) error codes. Avoiding these errors by staying within the limits of the scalability targets is an important part of enhancing your application's performance.

For more information about scalability targets for the Queue service, see [Azure Storage scalability and performance targets](#).

### Maximum number of storage accounts

If you're approaching the maximum number of storage accounts permitted for a particular subscription/region combination, evaluate your scenario and determine whether any of the following conditions apply:

- Are you using storage accounts to store unmanaged disks and adding those disks to your virtual machines (VMs)? For this scenario, Microsoft recommends using managed disks. Managed disks scale for you automatically and without the need to create and manage individual storage accounts. For more information, see [Introduction to Azure managed disks](#)
- Are you using one storage account per customer, for the purpose of data isolation? For this scenario, Microsoft recommends using a blob container for each customer, instead of an entire storage account. Azure Storage now allows you to assign Azure roles on a per-container basis. For more information, see [Assign an Azure role for access to blob data](#).
- Are you using multiple storage accounts to shard to increase ingress, egress, I/O operations per second (IOPS), or capacity? In this scenario, Microsoft recommends that you take advantage of increased limits for storage accounts to reduce the number of storage accounts required for your workload if possible. Contact [Azure Support](#) to request increased limits for your storage account. For more information, see [Announcing larger, higher scale storage accounts](#).

### Capacity and transaction targets

If your application is approaching the scalability targets for a single storage account, consider adopting one of the following approaches:

- If your application hits the transaction target, consider using block blob storage accounts, which are optimized for high transaction rates and low and consistent latency. For more information, see [Azure storage account overview](#).
- Reconsider the workload that causes your application to approach or exceed the scalability target. Can you design it differently to use less bandwidth or capacity, or fewer transactions?
- If your application must exceed one of the scalability targets, then create multiple storage accounts and partition your application data across those multiple storage accounts. If you use this pattern, then be sure to design your application so that you can add more storage accounts in the future for load balancing. Storage accounts themselves have no cost other than your usage in terms of data stored, transactions made, or data transferred.
- If your application is approaching the bandwidth targets, consider compressing data on the client side to reduce the bandwidth required to send the data to Azure Storage. While compressing data may save

bandwidth and improve network performance, it can also have negative effects on performance. Evaluate the performance impact of the additional processing requirements for data compression and decompression on the client side. Keep in mind that storing compressed data can make troubleshooting more difficult because it may be more challenging to view the data using standard tools.

- If your application is approaching the scalability targets, then make sure that you are using an exponential backoff for retries. It's best to try to avoid reaching the scalability targets by implementing the recommendations described in this article. However, using an exponential backoff for retries will prevent your application from retrying rapidly, which could make throttling worse. For more information, see the section titled [Timeout and Server Busy errors](#).

### Multiple clients accessing a single blob concurrently

If you have a large number of clients accessing a single blob concurrently, you will need to consider both per blob and per storage account scalability targets. The exact number of clients that can access a single blob will vary depending on factors such as the number of clients requesting the blob simultaneously, the size of the blob, and network conditions.

If the blob can be distributed through a CDN such as images or videos served from a website, then you can use a CDN. For more information, see the section titled [Content distribution](#).

In other scenarios, such as scientific simulations where the data is confidential, you have two options. The first is to stagger your workload's access such that the blob is accessed over a period of time vs being accessed simultaneously. Alternatively, you can temporarily copy the blob to multiple storage accounts to increase the total IOPS per blob and across storage accounts. Results will vary depending on your application's behavior, so be sure to test concurrency patterns during design.

### Bandwidth and operations per blob

A single blob supports up to 500 requests per second. If you have multiple clients that need to read the same blob and you might exceed this limit, then consider using a block blob storage account. A block blob storage account provides a higher request rate, or I/O operations per second (IOPS).

You can also use a content delivery network (CDN) such as Azure CDN to distribute operations on the blob. For more information about Azure CDN, see [Azure CDN overview](#).

## Partitioning

Understanding how Azure Storage partitions your blob data is useful for enhancing performance. Azure Storage can serve data in a single partition more quickly than data that spans multiple partitions. By naming your blobs appropriately, you can improve the efficiency of read requests.

Blob storage uses a range-based partitioning scheme for scaling and load balancing. Each blob has a partition key comprised of the full blob name (account+container+blob). The partition key is used to partition blob data into ranges. The ranges are then load-balanced across Blob storage.

Range-based partitioning means that naming conventions that use lexical ordering (for example, *mypayroll*, *myperformance*, *myemployees*, etc.) or timestamps (*log20160101*, *log20160102*, *log20160102*, etc.) are more likely to result in the partitions being co-located on the same partition server until increased load requires that they are split into smaller ranges. Co-locating blobs on the same partition server enhances performance, so an important part of performance enhancement involves naming blobs in a way that organizes them most effectively.

For example, all blobs within a container can be served by a single server until the load on these blobs requires further rebalancing of the partition ranges. Similarly, a group of lightly loaded accounts with their names arranged in lexical order may be served by a single server until the load on one or all of these accounts require them to be split across multiple partition servers.

Each load-balancing operation may impact the latency of storage calls during the operation. The service's ability

to handle a sudden burst of traffic to a partition is limited by the scalability of a single partition server until the load-balancing operation kicks in and rebalances the partition key range.

You can follow some best practices to reduce the frequency of such operations.

- If possible, use blob or block sizes greater than 256 KiB for standard and premium storage accounts. Larger blob or block sizes automatically activate high-throughput block blobs. High-throughput block blobs provide high-performance ingest that is not affected by partition naming.
- Examine the naming convention you use for accounts, containers, blobs, tables, and queues. Consider prefixing account, container, or blob names with a three-digit hash using a hashing function that best suits your needs.
- If you organize your data using timestamps or numerical identifiers, make sure that you are not using an append-only (or prepend-only) traffic pattern. These patterns are not suitable for a range-based partitioning system. These patterns may lead to all traffic going to a single partition and limiting the system from effectively load balancing.

For example, if you have daily operations that use a blob with a timestamp such as *yyyymmdd*, then all traffic for that daily operation is directed to a single blob, which is served by a single partition server. Consider whether the per-blob limits and per-partition limits meet your needs, and consider breaking this operation into multiple blobs if needed. Similarly, if you store time series data in your tables, all traffic may be directed to the last part of the key namespace. If you are using numerical IDs, prefix the ID with a three-digit hash. If you are using timestamps, prefix the timestamp with the seconds value, for example, *ssyyyymmdd*. If your application routinely performs listing and querying operations, choose a hashing function that will limit your number of queries. In some cases, a random prefix may be sufficient.

- For more information on the partitioning scheme used in Azure Storage, see [Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency](#).

## Networking

The physical network constraints of the application may have a significant impact on performance. The following sections describe some of limitations users may encounter.

### Client network capability

Bandwidth and the quality of the network link play important roles in application performance, as described in the following sections.

#### Throughput

For bandwidth, the problem is often the capabilities of the client. Larger Azure instances have NICs with greater capacity, so you should consider using a larger instance or more VMs if you need higher network limits from a single machine. If you are accessing Azure Storage from an on premises application, then the same rule applies: understand the network capabilities of the client device and the network connectivity to the Azure Storage location and either improve them as needed or design your application to work within their capabilities.

#### Link quality

As with any network usage, keep in mind that network conditions resulting in errors and packet loss will slow effective throughput. Using WireShark or NetMon may help in diagnosing this issue.

#### Location

In any distributed environment, placing the client near to the server delivers in the best performance. For accessing Azure Storage with the lowest latency, the best location for your client is within the same Azure region. For example, if you have an Azure web app that uses Azure Storage, then locate them both within a single region, such as US West or Asia Southeast. Co-locating resources reduces the latency and the cost, as bandwidth usage within a single region is free.

If client applications will access Azure Storage but are not hosted within Azure, such as mobile device apps or on-premises enterprise services, then locating the storage account in a region near to those clients may reduce latency. If your clients are broadly distributed (for example, some in North America, and some in Europe), then consider using one storage account per region. This approach is easier to implement if the data the application stores is specific to individual users, and does not require replicating data between storage accounts.

For broad distribution of blob content, use a content delivery network such as Azure CDN. For more information about Azure CDN, see [Azure CDN](#).

## SAS and CORS

Suppose that you need to authorize code such as JavaScript that is running in a user's web browser or in a mobile phone app to access data in Azure Storage. One approach is to build a service application that acts as a proxy. The user's device authenticates with the service, which in turn authorizes access to Azure Storage resources. In this way, you can avoid exposing your storage account keys on insecure devices. However, this approach places a significant overhead on the service application, because all of the data transferred between the user's device and Azure Storage must pass through the service application.

You can avoid using a service application as a proxy for Azure Storage by using shared access signatures (SAS). Using SAS, you can enable your user's device to make requests directly to Azure Storage by using a limited access token. For example, if a user wants to upload a photo to your application, then your service application can generate a SAS and send it to the user's device. The SAS token can grant permission to write to an Azure Storage resource for a specified interval of time, after which the SAS token expires. For more information about SAS, see [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#).

Typically, a web browser will not allow JavaScript in a page that is hosted by a website on one domain to perform certain operations, such as write operations, to another domain. Known as the same-origin policy, this policy prevents a malicious script on one page from obtaining access to data on another web page. However, the same-origin policy can be a limitation when building a solution in the cloud. Cross-origin resource sharing (CORS) is a browser feature that enables the target domain to communicate to the browser that it trusts requests originating in the source domain.

For example, suppose a web application running in Azure makes a request for a resource to an Azure Storage account. The web application is the source domain, and the storage account is the target domain. You can configure CORS for any of the Azure Storage services to communicate to the web browser that requests from the source domain are trusted by Azure Storage. For more information about CORS, see [Cross-origin resource sharing \(CORS\) support for Azure Storage](#).

Both SAS and CORS can help you avoid unnecessary load on your web application.

## Caching

Caching plays an important role in performance. The following sections discuss caching best practices.

### Reading data

In general, reading data once is preferable to reading it twice. Consider the example of a web application that has retrieved a 50 MiB blob from the Azure Storage to serve as content to a user. Ideally, the application caches the blob locally to disk and then retrieves the cached version for subsequent user requests.

One way to avoid retrieving a blob if it hasn't been modified since it was cached is to qualify the GET operation with a conditional header for modification time. If the last modified time is after the time that the blob was cached, then the blob is retrieved and re-cached. Otherwise, the cached blob is retrieved for optimal performance.

You may also decide to design your application to assume that the blob remains unchanged for a short period after retrieving it. In this case, the application does not need to check whether the blob was modified during that

interval.

Configuration data, lookup data, and other data that is frequently used by the application are good candidates for caching.

For more information about using conditional headers, see [Specifying conditional headers for Blob service operations](#).

### Uploading data in batches

In some scenarios, you can aggregate data locally, and then periodically upload it in a batch instead of uploading each piece of data immediately. For example, suppose a web application keeps a log file of activities. The application can either upload details of every activity as it happens to a table (which requires many storage operations), or it can save activity details to a local log file and then periodically upload all activity details as a delimited file to a blob. If each log entry is 1 KB in size, you can upload thousands of entries in a single transaction. A single transaction supports uploading a blob of up to 64 MiB in size. The application developer must design for the possibility of client device or upload failures. If the activity data needs to be downloaded for an interval of time rather than for a single activity, then using Blob storage is recommended over Table storage.

## .NET configuration

If using the .NET Framework, this section lists several quick configuration settings that you can use to make significant performance improvements. If using other languages, check to see if similar concepts apply in your chosen language.

### Use .NET Core

Develop your Azure Storage applications with .NET Core 2.1 or later to take advantage of performance enhancements. Using .NET Core 3.x is recommended when possible.

For more information on performance improvements in .NET Core, see the following blog posts:

- [Performance Improvements in .NET Core 3.0](#)
- [Performance Improvements in .NET Core 2.1](#)

### Increase default connection limit

In .NET, the following code increases the default connection limit (which is usually two in a client environment or ten in a server environment) to 100. Typically, you should set the value to approximately the number of threads used by your application. Set the connection limit before opening any connections.

```
ServicePointManager.DefaultConnectionLimit = 100; // (Or More)
```

For other programming languages, see the documentation to determine how to set the connection limit.

For more information, see the blog post [Web Services: Concurrent Connections](#).

### Increase minimum number of threads

If you are using synchronous calls together with asynchronous tasks, you may want to increase the number of threads in the thread pool:

```
ThreadPool.SetMinThreads(100,100); // (Determine the right number for your application)
```

For more information, see the [ThreadPool.SetMinThreads](#) method.

## Unbounded parallelism

While parallelism can be great for performance, be careful about using unbounded parallelism, meaning that

there is no limit enforced on the number of threads or parallel requests. Be sure to limit parallel requests to upload or download data, to access multiple partitions in the same storage account, or to access multiple items in the same partition. If parallelism is unbounded, your application can exceed the client device's capabilities or the storage account's scalability targets, resulting in longer latencies and throttling.

## Client libraries and tools

For best performance, always use the latest client libraries and tools provided by Microsoft. Azure Storage client libraries are available for a variety of languages. Azure Storage also supports PowerShell and Azure CLI. Microsoft actively develops these client libraries and tools with performance in mind, keeps them up-to-date with the latest service versions, and ensures that they handle many of the proven performance practices internally.

### TIP

The ABFS driver was designed to overcome the inherent deficiencies of WASB. Microsoft recommends using the ABFS driver over the WASB driver, as the ABFS driver is optimized specifically for big data analytics.

## Handle service errors

Azure Storage returns an error when the service cannot process a request. Understanding the errors that may be returned by Azure Storage in a given scenario is helpful for optimizing performance.

### Timeout and Server Busy errors

Azure Storage may throttle your application if it approaches the scalability limits. In some cases, Azure Storage may be unable to handle a request due to some transient condition. In both cases, the service may return a 503 (Server Busy) or 500 (Timeout) error. These errors can also occur if the service is rebalancing data partitions to allow for higher throughput. The client application should typically retry the operation that causes one of these errors. However, if Azure Storage is throttling your application because it is exceeding scalability targets, or even if the service was unable to serve the request for some other reason, aggressive retries may make the problem worse. Using an exponential back off retry policy is recommended, and the client libraries default to this behavior. For example, your application may retry after 2 seconds, then 4 seconds, then 10 seconds, then 30 seconds, and then give up completely. In this way, your application significantly reduces its load on the service, rather than exacerbating behavior that could lead to throttling.

Connectivity errors can be retried immediately, because they are not the result of throttling and are expected to be transient.

### Non-retryable errors

The client libraries handle retries with an awareness of which errors can be retried and which cannot. However, if you are calling the Azure Storage REST API directly, there are some errors that you should not retry. For example, a 400 (Bad Request) error indicates that the client application sent a request that could not be processed because it was not in the expected form. Resending this request results the same response every time, so there is no point in retrying it. If you are calling the Azure Storage REST API directly, be aware of potential errors and whether they should be retried.

For more information on Azure Storage error codes, see [Status and error codes](#).

## Copying and moving blobs

Azure Storage provides a number of solutions for copying and moving blobs within a storage account, between storage accounts, and between on-premises systems and the cloud. This section describes some of these options in terms of their effects on performance. For information about efficiently transferring data to or from

Blob storage, see [Choose an Azure solution for data transfer](#).

## **Blob copy APIs**

To copy blobs across storage accounts, use the [Put Block From URL](#) operation. This operation copies data synchronously from any URL source into a block blob. Using the [Put Block From URL](#) operation can significantly reduce required bandwidth when you are migrating data across storage accounts. Because the copy operation takes place on the service side, you do not need to download and re-upload the data.

To copy data within the same storage account, use the [Copy Blob](#) operation. Copying data within the same storage account is typically completed quickly.

## **Use AzCopy**

The AzCopy command-line utility is a simple and efficient option for bulk transfer of blobs to, from, and across storage accounts. AzCopy is optimized for this scenario, and can achieve high transfer rates. AzCopy version 10 uses the [Put Block From URL](#) operation to copy blob data across storage accounts. For more information, see [Copy or move data to Azure Storage by using AzCopy v10](#).

## **Use Azure Data Box**

For importing large volumes of data into Blob storage, consider using the Azure Data Box family for offline transfers. Microsoft-supplied Data Box devices are a good choice for moving large amounts of data to Azure when you're limited by time, network availability, or costs. For more information, see the [Azure DataBox Documentation](#).

## **Content distribution**

Sometimes an application needs to serve the same content to many users (for example, a product demo video used in the home page of a website), located in either the same or multiple regions. In this scenario, use a Content Delivery Network (CDN) such as Azure CDN to distribute blob content geographically. Unlike an Azure Storage account that exists in a single region and that cannot deliver content with low latency to other regions, Azure CDN uses servers in multiple data centers around the world. Additionally, a CDN can typically support much higher egress limits than a single storage account.

For more information about Azure CDN, see [Azure CDN](#).

## **Use metadata**

The Blob service supports HEAD requests, which can include blob properties or metadata. For example, if your application needs the Exif (exchangeable image format) data from a photo, it can retrieve the photo and extract it. To save bandwidth and improve performance, your application can store the Exif data in the blob's metadata when the application uploads the photo. You can then retrieve the Exif data in metadata using only a HEAD request. Retrieving only metadata and not the full contents of the blob saves significant bandwidth and reduces the processing time required to extract the Exif data. Keep in mind that 8 KiB of metadata can be stored per blob.

## **Upload blobs quickly**

To upload blobs quickly, first determine whether you will be uploading one blob or many. Use the below guidance to determine the correct method to use depending on your scenario.

### **Upload one large blob quickly**

To upload a single large blob quickly, a client application can upload its blocks or pages in parallel, being mindful of the scalability targets for individual blobs and the storage account as a whole. The Azure Storage client libraries support uploading in parallel. For example, you can use the following properties to specify the number of concurrent requests permitted in .NET or Java. Client libraries for other supported languages provide similar options.

- For .NET, set the [BlobRequestOptions.ParallelOperationThreadCount](#) property.
- For Java/Android, call the [BlobRequestOptions.setConcurrentRequestCount\(final Integer concurrentRequestCount\)](#) method.

## Upload many blobs quickly

To upload many blobs quickly, upload blobs in parallel. Uploading in parallel is faster than uploading single blobs at a time with parallel block uploads because it spreads the upload across multiple partitions of the storage service. AzCopy performs uploads in parallel by default, and is recommended for this scenario. For more information, see [Get started with AzCopy](#).

## Choose the correct type of blob

Azure Storage supports block blobs, append blobs, and page blobs. For a given usage scenario, your choice of blob type will affect the performance and scalability of your solution.

Block blobs are appropriate when you want to upload large amounts of data efficiently. For example, a client application that uploads photos or video to Blob storage would target block blobs.

Append blobs are similar to block blobs in that they are composed of blocks. When you modify an append blob, blocks are added to the end of the blob only. Append blobs are useful for scenarios such as logging, when an application needs to add data to an existing blob.

Page blobs are appropriate if the application needs to perform random writes on the data. For example, Azure virtual machine disks are stored as page blobs. For more information, see [Understanding block blobs, append blobs, and page blobs](#).

## Next steps

- [Scalability and performance targets for Blob storage](#)
- [Scalability and performance targets for standard storage accounts](#)
- [Status and error codes](#)

# Latency in Blob storage

8/22/2022 • 4 minutes to read • [Edit Online](#)

Latency, sometimes referenced as response time, is the amount of time that an application must wait for a request to complete. Latency can directly affect an application's performance. Low latency is often important for scenarios with humans in the loop, such as conducting credit card transactions or loading web pages. Systems that need to process incoming events at high rates, such as telemetry logging or IoT events, also require low latency. This article describes how to understand and measure latency for operations on block blobs, and how to design your applications for low latency.

Azure Storage offers two different performance options for block blobs: premium and standard. Premium block blobs offer significantly lower and more consistent latency than standard block blobs via high-performance SSD disks. For more information, see [Premium performance block blob storage in Hot, Cool, and Archive access tiers for blob data](#).

## About Azure Storage latency

Azure Storage latency is related to request rates for Azure Storage operations. Request rates are also known as input/output operations per second (IOPS).

To calculate the request rate, first determine the length of time that each request takes to complete, then calculate how many requests can be processed per second. For example, assume that a request takes 50 milliseconds (ms) to complete. An application using one thread with one outstanding read or write operation should achieve 20 IOPS (1 second or 1000 ms / 50 ms per request). Theoretically, if the thread count is doubled to two, then the application should be able to achieve 40 IOPS. If the outstanding asynchronous read or write operations for each thread are doubled to two, then the application should be able to achieve 80 IOPS.

In practice, request rates do not always scale so linearly, due to overhead in the client from task scheduling, context switching, and so forth. On the service side, there can be variability in latency due to pressure on the Azure Storage system, differences in the storage media used, noise from other workloads, maintenance tasks, and other factors. Finally, the network connection between the client and the server may affect Azure Storage latency due to congestion, rerouting, or other disruptions.

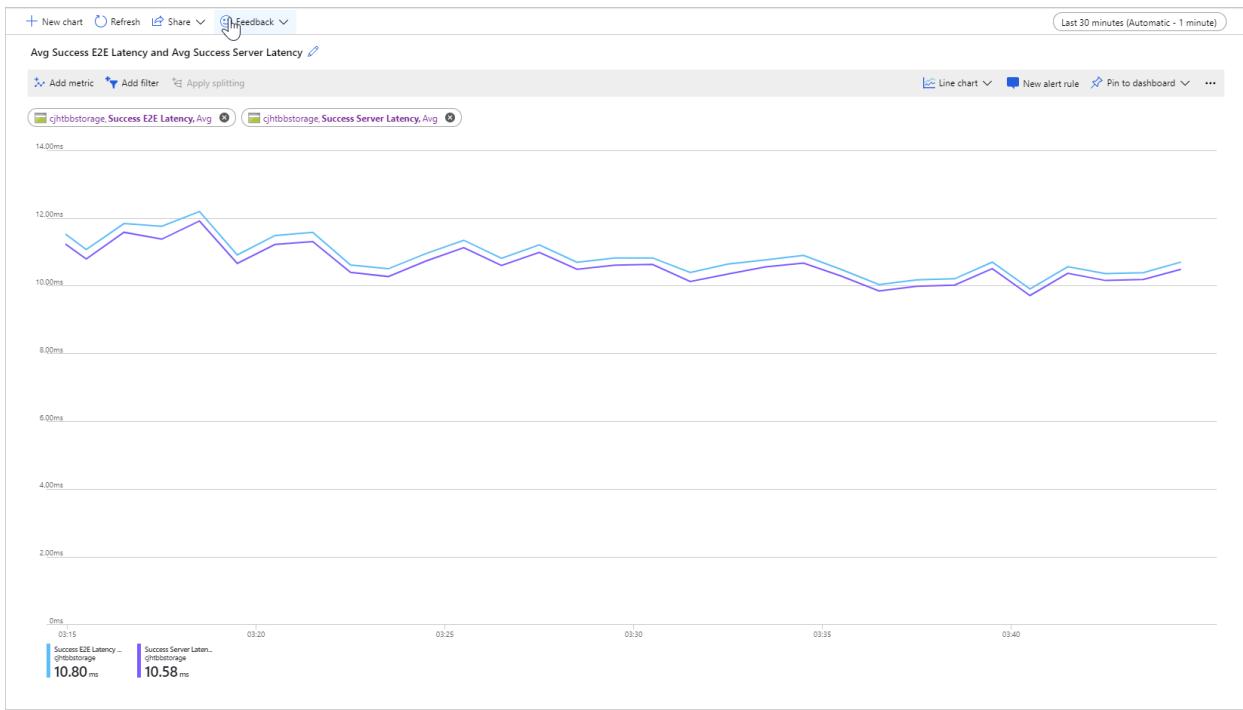
Azure Storage bandwidth, also referred to as throughput, is related to the request rate and can be calculated by multiplying the request rate (IOPS) by the request size. For example, assuming 160 requests per second, each 256 KiB of data results in throughput of 40,960 KiB per second or 40 MiB per second.

## Latency metrics for block blobs

Azure Storage provides two latency metrics for block blobs. These metrics can be viewed in the Azure portal:

- **End-to-end (E2E) latency** measures the interval from when Azure Storage receives the first packet of the request until Azure Storage receives a client acknowledgment on the last packet of the response.
- **Server latency** measures the interval from when Azure Storage receives the last packet of the request until the first packet of the response is returned from Azure Storage.

The following image shows the **Average Success E2E Latency** and **Average Success Server Latency** for a sample workload that calls the `Get Blob` operation:



Under normal conditions, there is little gap between end-to-end latency and server latency, which is what the image shows for the sample workload.

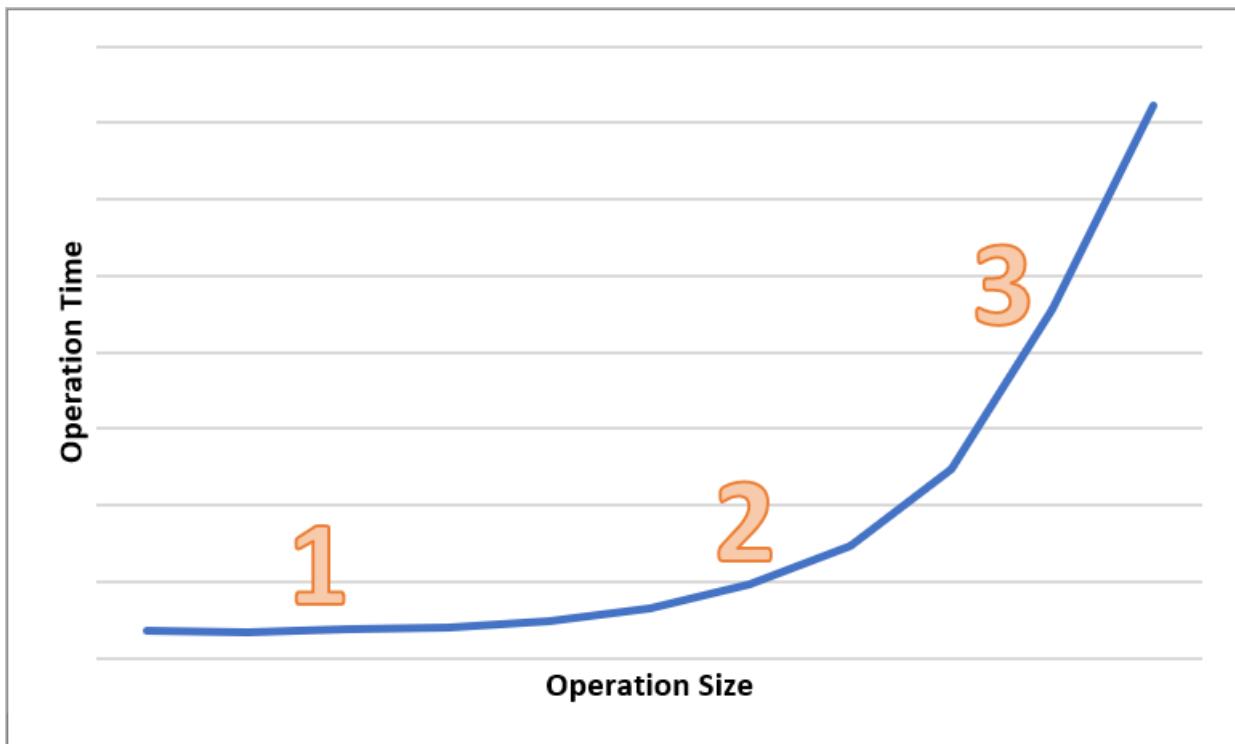
If you review your end-to-end and server latency metrics, and find that end-to-end latency is significantly higher than server latency, then investigate and address the source of the additional latency.

If your end-to-end and server latency are similar, but you require lower latency, then consider migrating to premium block blob storage.

## Factors influencing latency

The main factor influencing latency is operation size. It takes longer to complete larger operations, due to the amount of data being transferred over the network and processed by Azure Storage.

The following diagram shows the total time for operations of various sizes. For small amounts of data, the latency interval is predominantly spent handling the request, rather than transferring data. The latency interval increases only slightly as the operation size increases (marked 1 in the diagram below). As the operation size further increases, more time is spent on transferring data, so that the total latency interval is split between request handling and data transfer (marked 2 in the diagram below). With larger operation sizes, the latency interval is almost exclusively spent on transferring data and the request handling is largely insignificant (marked 3 in the diagram below).



Client configuration factors such as concurrency and threading also affect latency. Overall throughput depends on how many storage requests are in flight at any given point in time and on how your application handles threading. Client resources including CPU, memory, local storage, and network interfaces can also affect latency.

Processing Azure Storage requests requires client CPU and memory resources. If the client is under pressure due to an underpowered virtual machine or some runaway process in the system, there are fewer resources available to process Azure Storage requests. Any contention or lack of client resources will result in an increase in end-to-end latency without an increase in server latency, increasing the gap between the two metrics.

Equally important is the network interface and network pipe between the client and Azure Storage. Physical distance alone can be a significant factor, for example if a client VM is in a different Azure region or on-premises. Other factors such as network hops, ISP routing, and internet state can influence overall storage latency.

To assess latency, first establish baseline metrics for your scenario. Baseline metrics provide you with the expected end-to-end and server latency in the context of your application environment, depending on your workload profile, application configuration settings, client resources, network pipe, and other factors. When you have baseline metrics, you can more easily identify abnormal versus normal conditions. Baseline metrics also enable you to observe the effects of changed parameters, such as application configuration or VM sizes.

## Next steps

- [Scalability and performance targets for Blob storage](#)
- [Performance and scalability checklist for Blob storage](#)

# Optimize costs for Blob storage with reserved capacity

8/22/2022 • 8 minutes to read • [Edit Online](#)

You can save money on storage costs for blob data with Azure Storage reserved capacity. Azure Storage reserved capacity offers you a discount on capacity for block blobs and for Azure Data Lake Storage Gen2 data in standard storage accounts when you commit to a reservation for either one year or three years. A reservation provides a fixed amount of storage capacity for the term of the reservation.

Azure Storage reserved capacity can significantly reduce your capacity costs for block blobs and Azure Data Lake Storage Gen2 data. The cost savings achieved depend on the duration of your reservation, the total capacity you choose to reserve, and the access tier and type of redundancy that you've chosen for your storage account. Reserved capacity provides a billing discount and doesn't affect the state of your Azure Storage resources.

For information about Azure Storage reservation pricing, see [Block blob pricing](#) and [Azure Data Lake Storage Gen 2 pricing](#).

## Reservation terms for Azure Storage

The following sections describe the terms of an Azure Storage reservation.

### Reservation capacity

You can purchase Azure Storage reserved capacity in units of 100 TiB and 1 PiB per month for a one-year or three-year term.

### Reservation scope

Azure Storage reserved capacity is available for a single subscription, multiple subscriptions (shared scope), and management groups. When scoped to a single subscription, the reservation discount is applied to the selected subscription only. When scoped to multiple subscriptions, the reservation discount is shared across those subscriptions within the customer's billing context. When scoped to management group, the reservation discount is shared across the subscriptions that are a part of both the management group and billing scope.

When you purchase Azure Storage reserved capacity, you can use your reservation for both block blob and Azure Data Lake Storage Gen2 data. A reservation is applied to your usage within the purchased scope and cannot be limited to a specific storage account, container, or object within the subscription.

An Azure Storage reservation covers only the amount of data that is stored in a subscription or shared resource group. Early deletion, operations, bandwidth, and data transfer charges are not included in the reservation. As soon as you buy a reservation, the capacity charges that match the reservation attributes are charged at the discount rates instead of at the pay-as-you go rates. For more information on Azure reservations, see [What are Azure Reservations?](#)

### Supported account types, tiers, and redundancy options

Azure Storage reserved capacity is available for resources in standard storage accounts, including general-purpose v2 (GPv2) and Blob storage accounts.

All access tiers (hot, cool, and archive) are supported for reservations. For more information on access tiers, see [Hot, Cool, and Archive access tiers for blob data](#).

All types of redundancy are supported for reservations. For more information about redundancy options, see

**NOTE**

Azure Storage reserved capacity is not available for premium storage accounts, general-purpose v1 (GPv1) storage accounts, Azure Data Lake Storage Gen1, page blobs, Azure Queue storage, or Azure Table storage. For information about reserved capacity for Azure Files, see [Optimize costs for Azure Files with reserved capacity](#).

## Security requirements for purchase

To purchase reserved capacity:

- You must be in the **Owner** role for at least one Enterprise or individual subscription with pay-as-you-go rates.
- For Enterprise subscriptions, **Add Reserved Instances** must be enabled in the EA portal. Or, if that setting is disabled, you must be an EA Admin on the subscription.
- For the Cloud Solution Provider (CSP) program, only admin agents or sales agents can buy Azure Blob Storage reserved capacity.

## Determine required capacity before purchase

When you purchase an Azure Storage reservation, you must choose the region, access tier, and redundancy option for the reservation. Your reservation is valid only for data stored in that region, access tier, and redundancy level. For example, suppose you purchase a reservation for data in US West for the hot tier using zone-redundant storage (ZRS). You cannot use the same reservation for data in US East, data in the archive tier, or data in geo-redundant storage (GRS). However, you can purchase another reservation for your additional needs.

Reservations are available today for 100 TiB or 1 PiB blocks, with higher discounts for 1 PiB blocks. When you purchase a reservation in the Azure portal, Microsoft may provide you with recommendations based on your previous usage to help determine which reservation you should purchase.

## Purchase Azure Storage reserved capacity

You can purchase Azure Storage reserved capacity through the [Azure portal](#). Pay for the reservation up front or with monthly payments. For more information about purchasing with monthly payments, see [Purchase Azure reservations with up front or monthly payments](#).

For help with identifying the reservation terms that are right for your scenario, see [Understand the Azure Storage reserved capacity discount](#).

Follow these steps to purchase reserved capacity:

1. Navigate to the [Purchase reservations](#) pane in the Azure portal.
2. Select **Azure Blob Storage** to buy a new reservation.
3. Fill in the required fields as described in the following table:

Select the product you want to purchase

Save on your Azure data storage cost by pre-purchasing reserved capacity for 1 or 3 years. The reservation discount will automatically apply to data stored on Azure Blob (GPv2) and Azure Data Lake Storage (Gen 2). Discounts are applied hourly on the total data stored in that hour, unused reserved capacity doesn't carry over. [Learn More](#)

Scope \* Shared Subscription \* Sub for RI Testing

Filter by name... Region : West US Access tier : Hot Redundancy : ZRS Size : Select a value Reset filters

Billing frequency : Monthly Add Filter

Access tier	Redundancy	Size	Region	Term	Billing frequency
Hot	ZRS	100 Tb	West US	One Year	Monthly
Hot	ZRS	100 Tb	West US	Three Years	Monthly
Hot	ZRS	1 Pb	West US	One Year	Monthly
Hot	ZRS	1 Pb	West US	Three Years	Monthly

**Select** **Cancel** Monthly price per SKU: <price> USD  
% Estimated savings

FIELD	DESCRIPTION
Scope	<p>Indicates how many subscriptions can use the billing benefit associated with the reservation. It also controls how the reservation is applied to specific subscriptions.</p> <p>If you select <b>Shared</b>, the reservation discount is applied to Azure Storage capacity in any subscription within your billing context. The billing context is based on how you signed up for Azure. For enterprise customers, the shared scope is the enrollment and includes all subscriptions within the enrollment. For pay-as-you-go customers, the shared scope includes all individual subscriptions with pay-as-you-go rates created by the account administrator.</p> <p>If you select <b>Single subscription</b>, the reservation discount is applied to Azure Storage capacity in the selected subscription.</p> <p>If you select <b>Single resource group</b>, the reservation discount is applied to Azure Storage capacity in the selected subscription and the selected resource group within that subscription.</p> <p>You can change the reservation scope after you purchase the reservation.</p>
Subscription	<p>The subscription that's used to pay for the Azure Storage reservation. The payment method on the selected subscription is used in charging the costs. The subscription must be one of the following types:</p> <p>Enterprise Agreement (offer numbers: MS-AZR-0017P or MS-AZR-0148P): For an Enterprise subscription, the charges are deducted from the enrollment's Azure Prepayment (previously called monetary commitment) balance or charged as overage.</p> <p>Individual subscription with pay-as-you-go rates (offer numbers: MS-AZR-0003P or MS-AZR-0023P): For an individual subscription with pay-as-you-go rates, the charges are billed to the credit card or invoice payment method on the subscription.</p>

FIELD	DESCRIPTION
Region	The region where the reservation is in effect.
Access tier	The access tier where the reservation is in effect. Options include <i>Hot</i> , <i>Cool</i> , or <i>Archive</i> . For more information about access tiers, see <a href="#">Hot, Cool, and Archive access tiers for blob data</a> .
Redundancy	The redundancy option for the reservation. Options include <i>LRS</i> , <i>ZRS</i> , <i>GRS</i> , <i>GZRS</i> , <i>RA-GRS</i> , and <i>RA-GZRS</i> . For more information about redundancy options, see <a href="#">Azure Storage redundancy</a> .
Billing frequency	Indicates how often the account is billed for the reservation. Options include <i>Monthly</i> or <i>Upfront</i> .
Size	The amount of capacity to reserve.
Term	One year or three years.

4. After you select the parameters for your reservation, the Azure portal displays the cost. The portal also shows the discount percentage over pay-as-you-go billing.

5. In the **Purchase reservations** pane, review the total cost of the reservation. You can also provide a name for the reservation.

Reservation name	Product	Scope	Unit price	Quantity	Subtotal (% Discount)	Billing frequency
sample-reservation	Hot   ZRS   100 Tb   West US   Three Y...	Shared	<price>	1	<subtotal>	Monthly

Today's charge: <current-charge> USD  
View full payment schedule  
Total reservation cost: <total-cost> USD

After you purchase a reservation, it is automatically applied to any existing Azure Storage block blob or Azure Data Lake Storage Gen2 resources that matches the terms of the reservation. If you haven't created any Azure Storage resources yet, the reservation will apply whenever you create a resource that matches the terms of the reservation. In either case, the term of the reservation begins immediately after a successful purchase.

## Exchange or refund a reservation

You can exchange or refund a reservation, with certain limitations. These limitations are described in the following sections.

To exchange or refund a reservation, navigate to the reservation details in the Azure portal. Select **Exchange** or **Refund**, and follow the instructions to submit a support request. When the request has been processed, Microsoft will send you an email to confirm completion of the request.

For more information about Azure Reservations policies, see [Self-service exchanges and refunds for Azure](#)

[Reservations](#).

## Exchange a reservation

Exchanging a reservation enables you to receive a prorated refund based on the unused portion of the reservation. You can then apply the refund to the purchase price of a new Azure Storage reservation.

There's no limit on the number of exchanges you can make. Additionally, there's no fee associated with an exchange. The new reservation that you purchase must be of equal or greater value than the prorated credit from the original reservation. An Azure Storage reservation can be exchanged only for another Azure Storage reservation, and not for a reservation for any other Azure service.

## Refund a reservation

You may cancel an Azure Storage reservation at any time. When you cancel, you'll receive a prorated refund based on the remaining term of the reservation. The maximum refund per year is \$50,000.

Cancelling a reservation immediately terminates the reservation and returns the remaining months to Microsoft. The remaining prorated balance, minus the fee, will be refunded to your original form of purchase.

## Expiration of a reservation

When a reservation expires, any Azure Storage capacity that you are using under that reservation is billed at the pay-as-you go rate. Reservations don't renew automatically.

You will receive an email notification 30 days prior to the expiration of the reservation, and again on the expiration date. To continue taking advantage of the cost savings that a reservation provides, renew it no later than the expiration date.

## Need help? Contact us

If you have questions or need help, [create a support request](#).

## Next steps

- [What are Azure Reservations?](#)
- [Understand how the reservation discount is applied to Azure Storage](#)

# Plan and manage costs for Azure Blob Storage

8/22/2022 • 8 minutes to read • [Edit Online](#)

This article helps you plan and manage costs for Azure Blob Storage. First, estimate costs by using the Azure pricing calculator. After you create your storage account, optimize the account so that you pay only for what you need. Use cost management features to set budgets and monitor costs. You can also review forecasted costs, and monitor spending trends to identify areas where you might want to act.

Keep in mind that costs for Blob Storage are only a portion of the monthly costs in your Azure bill. Although this article explains how to estimate and manage costs for Blob Storage, you're billed for all Azure services and resources used for your Azure subscription, including the third-party services. After you're familiar with managing costs for Blob Storage, you can apply similar methods to manage costs for all the Azure services used in your subscription.

## Estimate costs

Use the [Azure pricing calculator](#) to estimate costs before you create and begin transferring data to an Azure Storage account.

1. On the [Azure pricing calculator](#) page, choose the **Storage Accounts** tile.
2. Scroll down the page and locate the **Storage Accounts** section of your estimate.
3. Choose options from the drop-down lists.

As you modify the value of these drop-down lists, the cost estimate changes. That estimate appears in the upper corner as well as the bottom of the estimate.

The screenshot shows the Azure Pricing Calculator interface. At the top, it says "Your Estimate" with a value of "\$21.94". Below this, there's a "Storage Accounts" section with various configuration options: REGION (West US), TYPE (Block Blob Storage), PERFORMANCE TIER (Standard), STORAGE ACCOUNT TYPE (Blob Storage), REDUNDANCY (LRS), and ACCESS TIER (Hot). To the right of these options is a "More info" section with three links: "Pricing details", "Product details", and "Documentation". A red box highlights the "More info" section.

As you change the value of the **Type** drop-down list, other options that appear on this worksheet change as well. Use the links in the **More Info** section to learn more about what each option means and how these options affect the price of storage-related operations.

4. Modify the remaining options to see their affect on your estimate.

## Understand the full billing model for Azure Blob Storage

Azure Blob Storage runs on Azure infrastructure that accrues costs when you deploy new resources. It's important to understand that there could be other additional infrastructure costs that might accrue.

### How you're charged for Azure Blob Storage

When you create or use Blob Storage resources, you'll be charged for the following meters:

METER	UNIT
Data storage	Per GB / per month
Operations	Per transaction
Data transfer	Per GB
Metadata	Per GB / per month <sup>1</sup>
Blob index tags	Per tag <sup>2</sup>
Change feed	Per logged change <sup>2</sup>
Encryption scopes	Per month <sup>2</sup>
Query acceleration	Per GB scanned & Per GB returned

<sup>1</sup> Applies only to accounts that have a hierarchical namespace.

<sup>2</sup> Applies only if you enable the feature.

Data traffic might also incur networking costs. See the [Bandwidth pricing](#).

At the end of your billing cycle, the charges for each meter are summed. Your bill or invoice shows a section for all Azure Blob Storage costs. There's a separate line item for each meter.

Data storage and metadata are billed per GB on a monthly basis. For data and metadata stored for less than a month, you can estimate the impact on your monthly bill by calculating the cost of each GB per day. You can use a similar approach to estimating the cost of encryption scopes that are in use for less than a month. The number of days in any given month varies. Therefore, to obtain the best approximation of your costs in a given month, make sure to divide the monthly cost by the number of days that occur in that month.

### Finding the unit price for each meter

To find unit prices, open the correct pricing page. If you've enabled the hierarchical namespace feature on your account, see the [Azure Data Lake Storage Gen2 pricing](#) page. If you haven't enabled this feature, see the [Block blob pricing](#) page.

In the pricing page, apply the appropriate redundancy, region, and currency filters. Prices for each meter appear in a table. Prices differ based on other settings in your account such as data redundancy options, access tier and performance tier.

### Flat namespace accounts and transaction pricing

Clients can make a request by using either the Blob Storage endpoint or the Data Lake Storage endpoint of your account. To learn more about storage account endpoints, see [Storage account endpoints](#).

Transaction prices that appear in the [Block blob pricing](#) page apply only to requests that use the Blob Storage endpoint (For example: `https://<storage-account>.blob.core.windows.net`). The listed prices do not apply to requests that use the Data Lake Storage Gen2 endpoint (For example:

`https://<storage-account>.dfs.core.windows.net`). For the transaction price of those requests, open the [Azure Data Lake Storage Gen2 pricing](#) page and select the **Flat Namespace** option.

<b>File Structure</b> <input type="button" value="Flat Namespace"/>	<b>Redundancy:</b> <input type="button" value="LRS"/>	<b>Region:</b> <input type="button" value="West US"/>	<b>Currency:</b> <input type="button" value="US Dollar (\$)"/>
------------------------------------------------------------------------	----------------------------------------------------------	----------------------------------------------------------	-------------------------------------------------------------------

Requests to the Data Lake Storage Gen2 endpoint can originate from any of the following sources:

- Workloads that use the Azure Blob File System driver or [ABFS driver](#).
- REST calls that use the [Azure Data Lake Store REST API](#)
- Applications that use Data Lake Storage Gen2 APIs from an Azure Storage client library.

### Using Azure Prepayment with Azure Blob Storage

You can pay for Azure Blob Storage charges with your Azure Prepayment (previously called monetary commitment) credit. However, you can't use Azure Prepayment credit to pay for charges for third party products and services including those from the Azure Marketplace.

## Optimize costs

Consider using these options to reduce costs.

- Reserve storage capacity
- Organize data into access tiers
- Automatically move data between access tiers

This section covers each option in more detail.

#### Reserve storage capacity

You can save money on storage costs for blob data with Azure Storage reserved capacity. Azure Storage reserved capacity offers you a discount on capacity for block blobs and for Azure Data Lake Storage Gen2 data in standard storage accounts when you commit to a reservation for either one year or three years. A reservation provides a fixed amount of storage capacity for the term of the reservation. Azure Storage reserved capacity can significantly reduce your capacity costs for block blobs and Azure Data Lake Storage Gen2 data.

To learn more, see [Optimize costs for Blob Storage with reserved capacity](#).

#### Organize data into access tiers

You can reduce costs by placing blob data into the most cost effective access tiers. Choose from three tiers that are designed to optimize your costs around data use. For example, the *hot* tier has a higher storage cost but lower access cost. Therefore, if you plan to access data frequently, the hot tier might be the most cost-efficient choice. If you plan to access data less frequently, the *cold* or *archive* tier might make the most sense because it raises the cost of accessing data while reducing the cost of storing data.

To learn more, see [Hot, Cool, and Archive access tiers for blob data](#).

#### Automatically move data between access tiers

Use lifecycle management policies to periodically move data between tiers to save the most money. These policies can move data to by using rules that you specify. For example, you might create a rule that moves blobs to the archive tier if that blob hasn't been modified in 90 days. By creating policies that adjust the access tier of your data, you can design the least expensive storage options for your needs.

To learn more, see [Manage the Azure Blob Storage lifecycle](#)

## Create budgets

You can create [budgets](#) to manage costs and create alerts that automatically notify stakeholders of spending

anomalies and overspending risks. Alerts are based on spending compared to budget and cost thresholds. Budgets and alerts are created for Azure subscriptions and resource groups, so they're useful as part of an overall cost monitoring strategy. However, they might have limited functionality to manage individual Azure service costs like the cost of Azure Storage because they are designed to track costs at a higher level.

## Monitor costs

As you use Azure resources with Azure Storage, you incur costs. Resource usage unit costs vary by time intervals (seconds, minutes, hours, and days) or by unit usage (bytes, megabytes, and so on.) Costs are incurred as soon as usage of Azure Storage starts. You can see the costs in the [cost analysis](#) pane in the Azure portal.

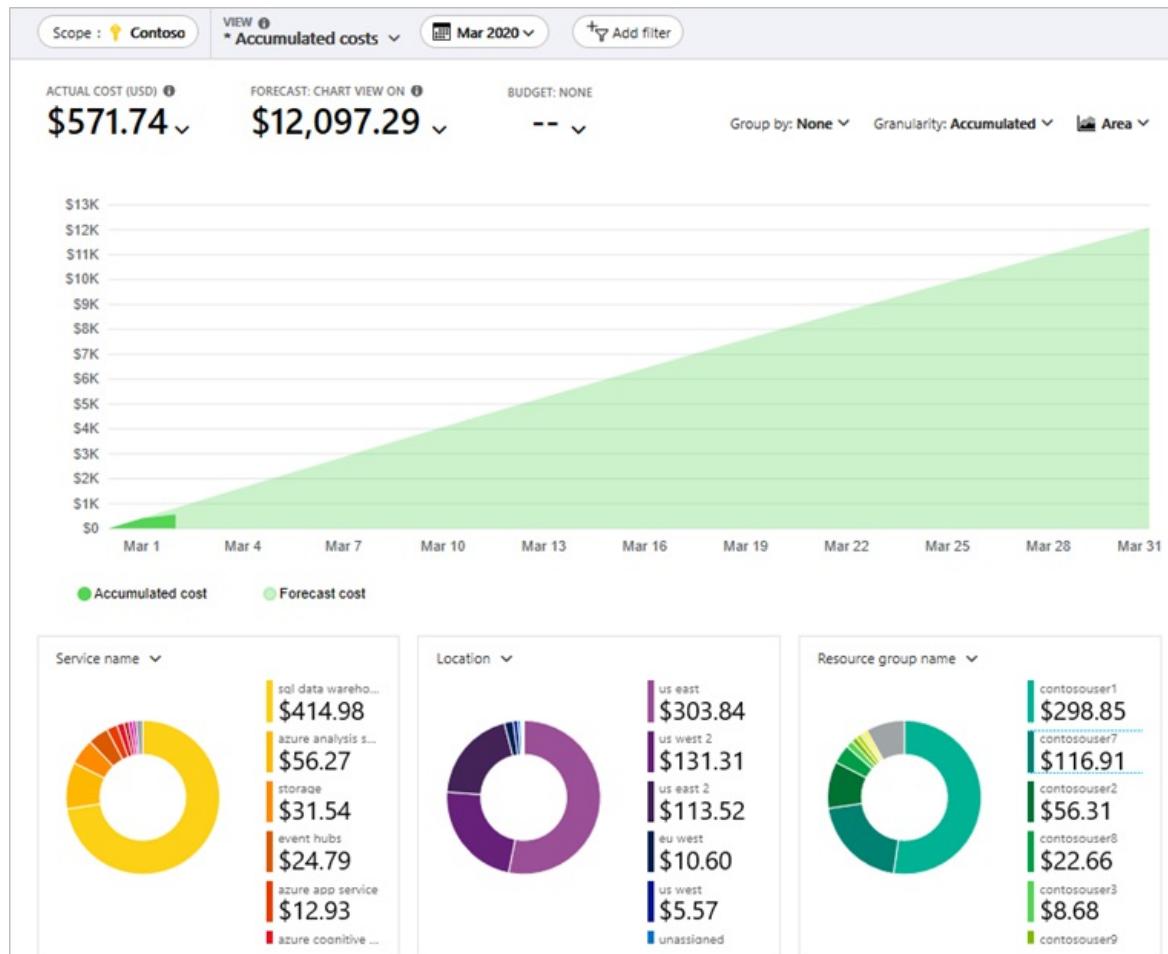
When you use cost analysis, you can view Azure Storage costs in graphs and tables for different time intervals. Some examples are by day, current and prior month, and year. You can also view costs against budgets and forecasted costs. Switching to longer views over time can help you identify spending trends and see where overspending might have occurred. If you've created budgets, you can also easily see where they exceeded.

### NOTE

Cost analysis supports different kinds of Azure account types. To view the full list of supported account types, see [Understand Cost Management data](#). To view cost data, you need at least read access for your Azure account. For information about assigning access to Azure Cost Management data, see [Assign access to data](#).

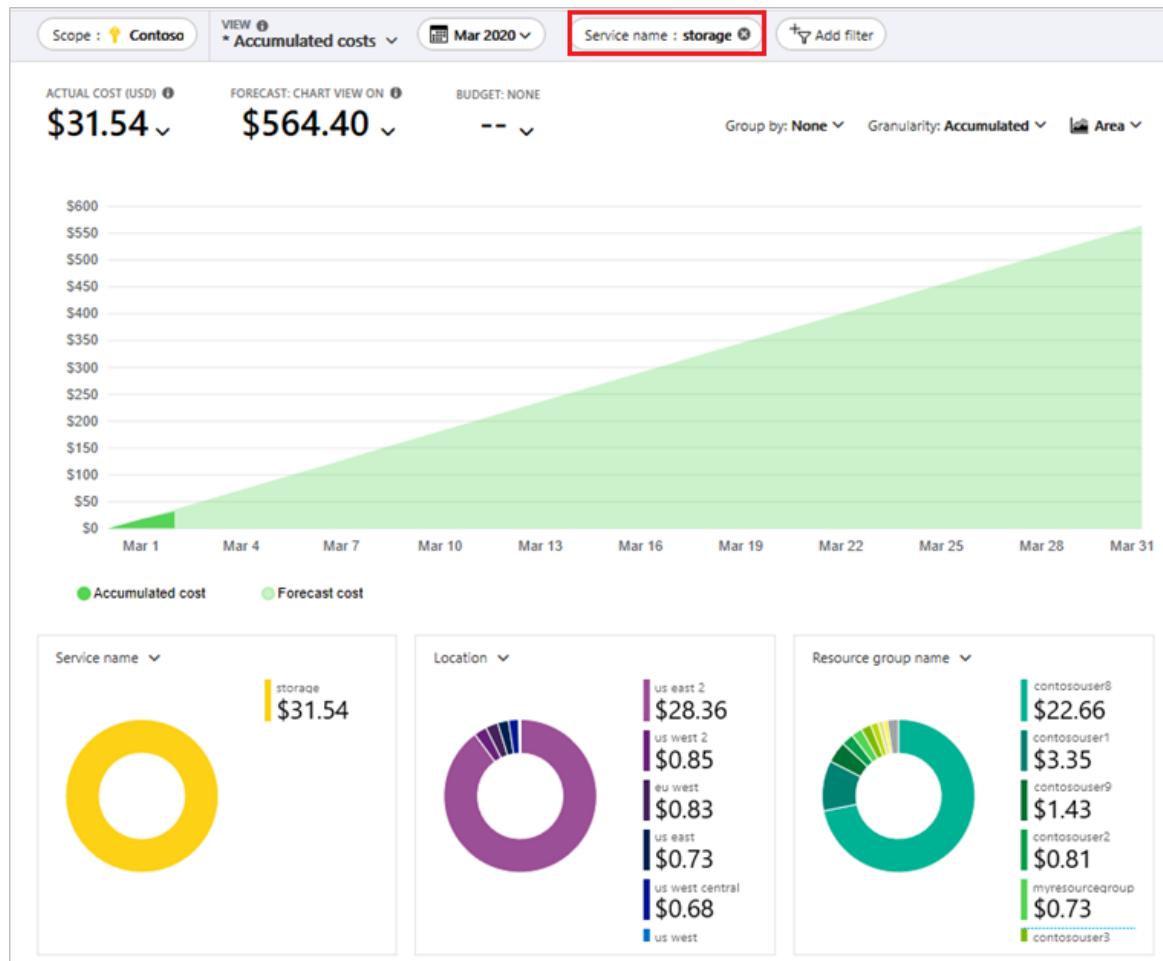
To view Azure Storage costs in cost analysis:

1. Sign into the [Azure portal](#).
2. Open the **Cost Management + Billing** window, select **Cost management** from the menu and then select **Cost analysis**. You can then change the scope for a specific subscription from the **Scope** dropdown.



3. To view only costs for Azure Storage, select **Add filter** and then select **Service name**. Then, choose **storage** from the list.

Here's an example showing costs for just Azure Storage:



In the preceding example, you see the current cost for the service. Costs by Azure regions (locations) and by resource group also appear. You can add other filters as well (For example: a filter to see costs for specific storage accounts).

## Export cost data

You can also [export your cost data](#) to a storage account. This is helpful when you need or others to do additional data analysis for costs. For example, a finance teams can analyze the data using Excel or Power BI. You can export your costs on a daily, weekly, or monthly schedule and set a custom date range. Exporting cost data is the recommended way to retrieve cost datasets.

## FAQ

### If I use Azure Storage for only a few days a month, is the cost prorated?

Storage capacity is billed in units of the average daily amount of data stored, in gigabytes (GB), over a monthly period. For example, if you consistently used 10 GB of storage for the first half of the month, and none for the second half of the month, you would be billed for your average usage of 5 GB of storage.

## Next steps

- Learn more on how pricing works with Azure Storage. See [Azure Storage Overview pricing](#).
- [Optimize costs for Blob Storage with reserved capacity](#).
- Learn [how to optimize your cloud investment with Azure Cost Management](#).

- Learn more about managing costs with [cost analysis](#).
- Learn about how to [prevent unexpected costs](#).
- Take the [Cost Management](#) guided learning course.

# Scalability and performance targets for Blob storage

8/22/2022 • 2 minutes to read • [Edit Online](#)

This reference details scalability and performance targets for Azure Storage. The scalability and performance targets listed here are high-end targets, but are achievable. In all cases, the request rate and bandwidth achieved by your storage account depends upon the size of objects stored, the access patterns utilized, and the type of workload your application performs.

Make sure to test your service to determine whether its performance meets your requirements. If possible, avoid sudden spikes in the rate of traffic and ensure that traffic is well-distributed across partitions.

When your application reaches the limit of what a partition can handle for your workload, Azure Storage begins to return error code 503 (Server Busy) or error code 500 (Operation Timeout) responses. If 503 errors are occurring, consider modifying your application to use an exponential backoff policy for retries. The exponential backoff allows the load on the partition to decrease, and to ease out spikes in traffic to that partition.

The service-level agreement (SLA) for Azure Storage accounts is available at [SLA for Storage Accounts](#).

## Scale targets for Blob storage

RESOURCE	TARGET
Maximum size of single blob container	Same as maximum storage account capacity
Maximum number of blocks in a block blob or append blob	50,000 blocks
Maximum size of a block in a block blob	4000 MiB
Maximum size of a block blob	50,000 X 4000 MiB (approximately 190.7 TiB)
Maximum size of a block in an append blob	4 MiB
Maximum size of an append blob	50,000 x 4 MiB (approximately 195 GiB)
Maximum size of a page blob	8 TiB <sup>2</sup>
Maximum number of stored access policies per blob container	5
Target request rate for a single blob	Up to 500 requests per second
Target throughput for a single page blob	Up to 60 MiB per second <sup>2</sup>
Target throughput for a single block blob	Up to storage account ingress/egress limits <sup>1</sup>

<sup>1</sup> Throughput for a single blob depends on several factors, including, but not limited to: concurrency, request size, performance tier, speed of source for uploads, and destination for downloads. To take advantage of the performance enhancements of [high-throughput block blobs](#), upload larger blobs or blocks. Specifically, call the [Put Blob](#) or [Put Block](#) operation with a blob or block size that is greater than 4 MiB for standard storage accounts. For premium block blob or for Data Lake Storage Gen2 storage accounts, use a block or blob size that

is greater than 256 KiB.

<sup>2</sup> Page blobs are not yet supported in accounts that have the **Hierarchical namespace** setting on them.

The following table describes the maximum block and blob sizes permitted by service version.

Service Version	Maximum Block Size (via Put Block)	Maximum Blob Size (via Put Block List)	Maximum Blob Size via Single Write Operation (via Put Blob)
Version 2019-12-12 and later	4000 MiB	Approximately 190.7 TiB (4000 MiB X 50,000 blocks)	5000 MiB (preview)
Version 2016-05-31 through version 2019-07-07	100 MiB	Approximately 4.75 TiB (100 MiB X 50,000 blocks)	256 MiB
Versions prior to 2016-05-31	4 MiB	Approximately 195 GiB (4 MiB X 50,000 blocks)	64 MiB

## See also

- [Performance and scalability checklist for Blob storage](#)
- [Scalability targets for standard storage accounts](#)
- [Scalability targets for premium block blob storage accounts](#)
- [Scalability targets for the Azure Storage resource provider](#)
- [Azure subscription limits and quotas](#)

# Scalability and performance targets for standard storage accounts

8/22/2022 • 3 minutes to read • [Edit Online](#)

This reference details scalability and performance targets for Azure Storage. The scalability and performance targets listed here are high-end targets, but are achievable. In all cases, the request rate and bandwidth achieved by your storage account depends upon the size of objects stored, the access patterns utilized, and the type of workload your application performs.

Make sure to test your service to determine whether its performance meets your requirements. If possible, avoid sudden spikes in the rate of traffic and ensure that traffic is well-distributed across partitions.

When your application reaches the limit of what a partition can handle for your workload, Azure Storage begins to return error code 503 (Server Busy) or error code 500 (Operation Timeout) responses. If 503 errors are occurring, consider modifying your application to use an exponential backoff policy for retries. The exponential backoff allows the load on the partition to decrease, and to ease out spikes in traffic to that partition.

The service-level agreement (SLA) for Azure Storage accounts is available at [SLA for Storage Accounts](#).

## Scale targets for standard storage accounts

The following table describes default limits for Azure general-purpose v2 (GPv2), general-purpose v1 (GPv1), and Blob storage accounts. The *ingress* limit refers to all data that is sent to a storage account. The *egress* limit refers to all data that is received from a storage account.

Microsoft recommends that you use a GPv2 storage account for most scenarios. You can easily upgrade a GPv1 or a Blob storage account to a GPv2 account with no downtime and without the need to copy data. For more information, see [Upgrade to a GPv2 storage account](#).

### NOTE

You can request higher capacity and ingress limits. To request an increase, contact [Azure Support](#).

RESOURCE	LIMIT
Maximum number of storage accounts with standard endpoints per region per subscription, including standard and premium storage accounts.	250
Maximum number of storage accounts with Azure DNS zone endpoints (preview) per region per subscription, including standard and premium storage accounts.	5000 (preview)
Default maximum storage account capacity	5 PiB <sup>1</sup>
Maximum number of blob containers, blobs, file shares, tables, queues, entities, or messages per storage account.	No limit
Default maximum request rate per storage account	20,000 requests per second <sup>1</sup>

RESOURCE	LIMIT
<p>Default maximum ingress per general-purpose v2 and Blob storage account in the following regions (LRS/GRS):</p> <ul style="list-style-type: none"> <li>• Australia East</li> <li>• Central US</li> <li>• East Asia</li> <li>• East US 2</li> <li>• Japan East</li> <li>• Korea Central</li> <li>• North Europe</li> <li>• South Central US</li> <li>• Southeast Asia</li> <li>• UK South</li> <li>• West Europe</li> <li>• West US</li> </ul>	60 Gbps <sup>1</sup>
<p>Default maximum ingress per general-purpose v2 and Blob storage account in the following regions (ZRS):</p> <ul style="list-style-type: none"> <li>• Australia East</li> <li>• Central US</li> <li>• East US</li> <li>• East US 2</li> <li>• Japan East</li> <li>• North Europe</li> <li>• South Central US</li> <li>• Southeast Asia</li> <li>• UK South</li> <li>• West Europe</li> <li>• West US 2</li> </ul>	60 Gb ps <sup>1</sup>
Default maximum ingress per general-purpose v2 and Blob storage account in regions that aren't listed in the previous row.	25 Gbps <sup>1</sup>
Default maximum ingress for general-purpose v1 storage accounts (all regions)	10 Gbps <sup>1</sup>
<p>Default maximum egress for general-purpose v2 and Blob storage accounts in the following regions (LRS/GRS):</p> <ul style="list-style-type: none"> <li>• Australia East</li> <li>• Central US</li> <li>• East Asia</li> <li>• East US 2</li> <li>• Japan East</li> <li>• Korea Central</li> <li>• North Europe</li> <li>• South Central US</li> <li>• Southeast Asia</li> <li>• UK South</li> <li>• West Europe</li> <li>• West US</li> </ul>	120 Gbps <sup>1</sup>

RESOURCE	LIMIT
Default maximum egress for general-purpose v2 and Blob storage accounts in the following regions (ZRS): <ul style="list-style-type: none"> <li>• Australia East</li> <li>• Central US</li> <li>• East US</li> <li>• East US 2</li> <li>• Japan East</li> <li>• North Europe</li> <li>• South Central US</li> <li>• Southeast Asia</li> <li>• UK South</li> <li>• West Europe</li> <li>• West US 2</li> </ul>	120 Gbps <sup>1</sup>
Default maximum egress for general-purpose v2 and Blob storage accounts in regions that aren't listed in the previous row.	50 Gbps <sup>1</sup>
Maximum number of IP address rules per storage account	200
Maximum number of virtual network rules per storage account	200
Maximum number of resource instance rules per storage account	200
Maximum number of private endpoints per storage account	200

<sup>1</sup> Azure Storage standard accounts support higher capacity limits and higher limits for ingress and egress by request. To request an increase in account limits, contact [Azure Support](#).

## See also

- [Scalability targets for the Azure Storage resource provider](#)
- [Azure subscription limits and quotas](#)

# Scalability targets for premium block blob storage accounts

8/22/2022 • 2 minutes to read • [Edit Online](#)

A premium-performance block blob storage account is optimized for applications that use smaller, kilobyte-range objects. It's ideal for applications that require high transaction rates or consistent low-latency storage. Premium performance block blob storage is designed to scale with your applications. If your scenario requires that you deploy application(s) that require hundreds of thousands of requests per second or petabytes of storage capacity, contact Microsoft by submitting a support request in the [Azure portal](#).

The service-level agreement (SLA) for Azure Storage accounts is available at [SLA for Storage Accounts](#).

# Scalability and performance targets for premium page blob storage accounts

8/22/2022 • 2 minutes to read • [Edit Online](#)

This reference details scalability and performance targets for Azure Storage. The scalability and performance targets listed here are high-end targets, but are achievable. In all cases, the request rate and bandwidth achieved by your storage account depends upon the size of objects stored, the access patterns utilized, and the type of workload your application performs.

Make sure to test your service to determine whether its performance meets your requirements. If possible, avoid sudden spikes in the rate of traffic and ensure that traffic is well-distributed across partitions.

When your application reaches the limit of what a partition can handle for your workload, Azure Storage begins to return error code 503 (Server Busy) or error code 500 (Operation Timeout) responses. If 503 errors are occurring, consider modifying your application to use an exponential backoff policy for retries. The exponential backoff allows the load on the partition to decrease, and to ease out spikes in traffic to that partition.

The service-level agreement (SLA) for Azure Storage accounts is available at [SLA for Storage Accounts](#).

## Scale targets for premium page blob accounts

A premium-performance page blob storage account is optimized for read/write operations. This type of storage account backs an unmanaged disk for an Azure virtual machine.

### NOTE

Microsoft recommends using managed disks with Azure virtual machines (VMs) if possible. For more information about managed disks, see [Azure Disk Storage overview for VMs](#).

Premium page blob storage accounts have the following scalability targets:

TOTAL ACCOUNT CAPACITY	TOTAL BANDWIDTH FOR A LOCALLY REDUNDANT STORAGE ACCOUNT
Disk capacity: 4 TB (individual disk)/ 35 TB (cumulative total of all disks) Snapshot capacity: 10 TB <sup>3</sup>	Up to 50 gigabits per second for inbound <sup>1</sup> + outbound <sup>2</sup>

<sup>1</sup> All data (requests) that are sent to a storage account

<sup>2</sup> All data (responses) that are received from a storage account

<sup>3</sup> The total number of snapshots an individual page blob can have is 100.

A premium page blob account is a general-purpose account configured for premium performance. General-purpose v2 storage accounts are recommended.

If you are using premium page blob storage accounts for unmanaged disks and your application exceeds the scalability targets of a single storage account, then Microsoft recommends migrating to managed disks. For more information about managed disks, see [Azure Disk Storage overview for VMs](#).

If you cannot migrate to managed disks, then build your application to use multiple storage accounts and

partition your data across those storage accounts. For example, if you want to attach 51-TB disks across multiple VMs, spread them across two storage accounts. 35 TB is the limit for a single premium storage account. Make sure that a single premium performance storage account never has more than 35 TB of provisioned disks.

## See also

- [Scalability and performance targets for standard storage accounts](#)
- [Scalability targets for premium block blob storage accounts](#)
- [Azure subscription limits and quotas](#)

# Scalability and performance targets for the Azure Storage resource provider

8/22/2022 • 2 minutes to read • [Edit Online](#)

This reference details scalability and performance targets for Azure Storage. The scalability and performance targets listed here are high-end targets, but are achievable. In all cases, the request rate and bandwidth achieved by your storage account depends upon the size of objects stored, the access patterns utilized, and the type of workload your application performs.

Make sure to test your service to determine whether its performance meets your requirements. If possible, avoid sudden spikes in the rate of traffic and ensure that traffic is well-distributed across partitions.

When your application reaches the limit of what a partition can handle for your workload, Azure Storage begins to return error code 503 (Server Busy) or error code 500 (Operation Timeout) responses. If 503 errors are occurring, consider modifying your application to use an exponential backoff policy for retries. The exponential backoff allows the load on the partition to decrease, and to ease out spikes in traffic to that partition.

The service-level agreement (SLA) for Azure Storage accounts is available at [SLA for Storage Accounts](#).

## Scale targets for the resource provider

The following limits apply only when you perform management operations by using Azure Resource Manager with Azure Storage.

RESOURCE	LIMIT
Storage account management operations (read)	800 per 5 minutes
Storage account management operations (write)	10 per second / 1200 per hour
Storage account management operations (list)	100 per 5 minutes

## See also

- [Scalability and performance targets for standard storage accounts](#)
- [Azure subscription limits and quotas](#)

# Azure Storage blob inventory

8/22/2022 • 12 minutes to read • [Edit Online](#)

The Azure Storage blob inventory feature provides an overview of your containers, blobs, snapshots, and blob versions within a storage account. Use the inventory report to understand various attributes of blobs and containers such as your total data size, age, encryption status, immutability policy, and legal hold and so on. The report provides an overview of your data for business and compliance requirements.

## Inventory features

The following list describes features and capabilities that are available in the current release of Azure Storage blob inventory.

- **Inventory reports for blobs and containers**

You can generate inventory reports for blobs and containers. A report for blobs can contain base blobs, snapshots, blob versions and their associated properties such as creation time, last modified time. A report for containers describes containers and their associated properties such as immutability policy status, legal hold status.

- **Custom Schema**

You can choose which fields appear in reports. Choose from a list of supported fields. That list appears later in this article.

- **CSV and Apache Parquet output format**

You can generate an inventory report in either CSV or Apache Parquet output format.

- **Manifest file and Azure Event Grid event per inventory report**

A manifest file and an Azure Event Grid event are generated per inventory report. These are described later in this article.

## Enabling inventory reports

Enable blob inventory reports by adding a policy with one or more rules to your storage account. For guidance, see [Enable Azure Storage blob inventory reports](#).

## Upgrading an inventory policy

If you are an existing Azure Storage blob inventory user who has configured inventory prior to June 2021, you can start using the new features by loading the policy, and then saving the policy back after making changes. When you reload the policy, the new fields in the policy will be populated with default values. You can change these values if you want. Also, the following two features will be available.

- A destination container is now supported for every rule instead of just being supported for the policy.
- A manifest file and Azure Event Grid event are now generated per rule instead of per policy.

## Inventory policy

An inventory report is configured by adding an inventory policy with one or more rules. An inventory policy is a collection of rules in a JSON document.

```
{
 "enabled": true,
 "rules": [
 {
 "enabled": true,
 "name": "inventoryrule1",
 "destination": "inventory-destination-container",
 "definition": {. . .}
 },
 {
 "enabled": true,
 "name": "inventoryrule2",
 "destination": "inventory-destination-container",
 "definition": {. . .}
 }
]
}
```

View the JSON for an inventory policy by selecting the **Code view** tab in the **Blob inventory** section of the Azure portal.

PARAMETER NAME	PARAMETER TYPE	NOTES	REQUIRED?
enabled	boolean	Used to disable the entire policy. When set to <b>true</b> , the rule level enabled field overrides this parameter. When disabled, inventory for all rules will be disabled.	Yes
rules	Array of rule objects	At least one rule is required in a policy. Up to 100 rules are supported per policy.	Yes

## Inventory rules

A rule captures the filtering conditions and output parameters for generating an inventory report. Each rule creates an inventory report. Rules can have overlapping prefixes. A blob can appear in more than one inventory depending on rule definitions.

Each rule within the policy has several parameters:

PARAMETER NAME	PARAMETER TYPE	NOTES	REQUIRED?
name	string	A rule name can include up to 256 case-sensitive alphanumeric characters. The name must be unique within a policy.	Yes
enabled	boolean	A flag allowing a rule to be enabled or disabled. The default value is <b>true</b> .	Yes
definition	JSON inventory rule definition	Each definition is made up of a rule filter set.	Yes

PARAMETER NAME	PARAMETER TYPE	NOTES	REQUIRED?
destination	string	The destination container where all inventory files will be generated. The destination container must already exist.	

The global **Blob inventory enabled** flag takes precedence over the *enabled* parameter in a rule.

### Rule definition

PARAMETER NAME	PARAMETER TYPE	NOTES	REQUIRED
filters	json	Filters decide whether a blob or container is part of inventory or not.	Yes
format	string	Determines the output of the inventory file. Valid values are <code>csv</code> (For CSV format) and <code>parquet</code> (For Apache Parquet format).	Yes
objectType	string	Denotes whether this is an inventory rule for blobs or containers. Valid values are <code>blob</code> and <code>container</code> .	Yes
schedule	string	Schedule on which to run this rule. Valid values are <code>daily</code> and <code>weekly</code> .	Yes
schemaFields	Json array	List of Schema fields to be part of inventory.	Yes

### Rule filters

Several filters are available for customizing a blob inventory report:

FILTER NAME	FILTER TYPE	NOTES	REQUIRED?
blobTypes	Array of predefined enum values	Valid values are <code>blockBlob</code> and <code>appendBlob</code> for hierarchical namespace enabled accounts, and <code>blockBlob</code> , <code>appendBlob</code> , and <code>pageBlob</code> for other accounts. This field is not applicable for inventory on a container, (objectType: <code>container</code> ).	Yes

FILTER NAME	FILTER TYPE	NOTES	REQUIRED?
prefixMatch	Array of up to 10 strings for prefixes to be matched.	<p>If you don't define <i>prefixMatch</i> or provide an empty prefix, the rule applies to all blobs within the storage account. A prefix must be a container name prefix or a container name. For example,</p> <pre data-bbox="822 435 996 496"><code>container , container1/foo .</code></pre>	No
excludePrefix	Array of up to 10 strings for prefixes to be excluded.	<p>Specifies the blob paths to exclude from the inventory report.</p> <p>An <i>excludePrefix</i> must be a container name prefix or a container name. An empty <i>excludePrefix</i> would mean that all blobs with names matching any <i>prefixMatch</i> string will be listed.</p> <p>If you want to include a certain prefix, but exclude some specific subset from it, then you could use the <i>excludePrefix</i> filter. For example, if you want to include all blobs under <code>container-a</code> except those under the folder <code>container-a/folder</code>, then <i>prefixMatch</i> should be set to <code>container-a</code> and <i>excludePrefix</i> should be set to <code>container-a/folder</code>.</p>	No
includeSnapshots	boolean	<p>Specifies whether the inventory should include snapshots. Default is <code>false</code>. This field is not applicable for inventory on a container, (objectType: <code>container</code>).</p>	No
includeBlobVersions	boolean	<p>Specifies whether the inventory should include blob versions. Default is <code>false</code>. This field is not applicable for inventory on a container, (objectType: <code>container</code>).</p>	No

View the JSON for inventory rules by selecting the **Code view** tab in the **Blob inventory** section of the Azure portal. Filters are specified within a rule definition.

```
{
 "destination": "inventory-destination-container",
 "enabled": true,
 "rules": [
 {
 "definition": {
 "filters": {
 "blobTypes": ["blockBlob", "appendBlob", "pageBlob"],
 "prefixMatch": ["inventorytestcontainer1", "inventorytestcontainer2/abcd", "etc"],
 "excludePrefix": ["inventorytestcontainer10", "etc/logs"],
 "includeSnapshots": false,
 "includeBlobVersions": true,
 },
 "format": "csv",
 "objectType": "blob",
 "schedule": "daily",
 "schemaFields": ["Name", "Creation-Time"]
 },
 "enabled": true,
 "name": "blobinventorytest",
 "destination": "inventorydestinationContainer"
 },
 {
 "definition": {
 "filters": {
 "prefixMatch": ["inventorytestcontainer1", "inventorytestcontainer2/abcd", "etc"]
 },
 "format": "csv",
 "objectType": "container",
 "schedule": "weekly",
 "schemaFields": ["Name", "HasImmutabilityPolicy", "HasLegalHold"]
 },
 "enabled": true,
 "name": "containerinventorytest",
 "destination": "inventorydestinationContainer"
 }
]
}
```

## Custom schema fields supported for blob inventory

### NOTE

The Data Lake Storage Gen2 column shows support in accounts that have the hierarchical namespace feature enabled.

FIELD	BLOB STORAGE (DEFAULT SUPPORT)	DATA LAKE STORAGE GEN2
Name (Required)	✓	✓
Creation-Time	✓	✓
Last-Modified	✓	✓
Last-Access-Time	✓	✓
ETag	✓	✓
Content-Length	✓	✓

FIELD	BLOB STORAGE (DEFAULT SUPPORT)	DATA LAKE STORAGE GEN2
Content-Type	✓	✓
Content-Encoding	✓	✓
Content-Language	✓	✓
Content-CRC64	✓	✓
Content-MD5	✓	✓
Cache-Control	✓	✓
Cache-Disposition	✓	✓
BlobType	✓	✓
AccessTier	✓	✓
AccessTierChangeTime	✓	✓
LeaseStatus	✓	✓
LeaseState	✓	✓
ServerEncrypted	✓	✓
CustomerProvidedKeySHA256	✓	✓
Metadata	✓	✓
Expiry-Time	✗	✓
hdi_isfolder	✗	✓
Owner	✗	✓
Group	✗	✓
Permissions	✗	✓
Ad	✗	✓
Snapshot (Available and required when you choose to include snapshots in your report)	✓	✓
Deleted	✓	✓
DeletedId	✗	✓

FIELD	BLOB STORAGE (DEFAULT SUPPORT)	DATA LAKE STORAGE GEN2
DeletedTime	☒	✓
RemainingRetentionDays	✓	✓
VersionId (Available and required when you choose to include blob versions in your report)	✓	☒
IsCurrentVersion (Available and required when you choose to include blob versions in your report)	✓	☒
TagCount	✓	☒
Tags	✓	☒
CopyId	✓	✓
CopySource	✓	✓
CopyStatus	✓	✓
CopyProgress	✓	✓
CopyCompletionTime	✓	✓
CopyStatusDescription	✓	✓
ImmutabilityPolicyUntilDate	✓	✓
ImmutabilityPolicyMode	✓	✓
LegalHold	✓	✓
RehydratePriority	✓	✓
ArchiveStatus	✓	✓
EncryptionScope	✓	✓
IncrementalCopy	✓	✓
x-ms-blob-sequence-number	✓	☒

### Custom schema fields supported for container inventory

**NOTE**

The Data Lake Storage Gen2 column shows support in accounts that have the hierarchical namespace feature enabled.

FIELD	BLOB STORAGE (DEFAULT SUPPORT)	DATA LAKE STORAGE GEN2
Name (Required)	✓	✓
Last-Modified	✓	✓
ETag	✓	✓
LeaseStatus	✓	✓
LeaseState	✓	✓
LeaseDuration	✓	✓
Metadata	✓	✓
PublicAccess	✓	✓
DefaultEncryptionScope	✓	✓
DenyEncryptionScopeOverride	✓	✓
HasImmutabilityPolicy	✓	✓
HasLegalHold	✓	✓
ImmutableStorageWithVersioningEnabled	✓	✓
Deleted (Will appear only if include deleted containers is selected)	✓	✓
Version (Will appear only if include deleted containers is selected)	✓	✓
DeletedTime (Will appear only if include deleted containers is selected)	✓	✓
RemainingRetentionDays (Will appear only if include deleted containers is selected)	✓	✓

## Inventory run

A blob inventory run is automatically scheduled every day. It can take up to 24 hours for an inventory run to complete. For hierarchical namespace enabled accounts, a run can take as long as two days, and depending on the number of files being processed, the run might not complete by end of that two days. If a run does not complete successfully, check subsequent runs to see if they complete before contacting support. The performance of a run can vary, so if a run doesn't complete, it's possible that subsequent runs will.

Inventory policies are read or written in full. Partial updates aren't supported.

## IMPORTANT

If you enable firewall rules for your storage account, inventory requests might be blocked. You can unblock these requests by providing exceptions for trusted Microsoft services. For more information, see the Exceptions section in [Configure firewalls and virtual networks](#).

## Inventory completed event

The `BlobInventoryPolicyCompleted` event is generated when the inventory run completes for a rule. This event also occurs if the inventory run fails with a user error before it starts to run. For example, an invalid policy, or an error that occurs when a destination container is not present will trigger the event. The following json shows an example `BlobInventoryPolicyCompleted` event.

```
{
 "topic": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx/resourceGroups/BlobInventory/providers/Microsoft.EventGrid/topics/BlobInventoryTopic",
 "subject": "BlobDataManagement/BlobInventory",
 "eventType": "Microsoft.Storage.BlobInventoryPolicyCompleted",
 "id": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
 "data": {
 "scheduleDateTime": "2021-05-28T03:50:27Z",
 "accountName": "testaccount",
 "ruleName": "Rule_1",
 "policyRunStatus": "Succeeded",
 "policyRunStatusMessage": "Inventory run succeeded, refer manifest file for inventory details.",
 "policyRunId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
 "manifestBlobUrl": "https://testaccount.blob.core.windows.net/inventory-destination-
container/2021/05/26/13-25-36/Rule_1/Rule_1-manifest.json"
 },
 "dataVersion": "1.0",
 "metadataVersion": "1",
 "eventTime": "2021-05-28T15:03:18Z"
}
```

The following table describes the schema of the `BlobInventoryPolicyCompleted` event.

FIELD	TYPE	DESCRIPTION
scheduleDateTime	string	The time that the inventory policy was scheduled.
accountName	string	The storage account name.
ruleName	string	The rule name.
policyRunStatus	string	The status of inventory run. Possible values are <code>Succeeded</code> , <code>PartiallySucceeded</code> , and <code>Failed</code> .
policyRunStatusMessage	string	The status message for the inventory run.
policyRunId	string	The policy run ID for the inventory run.

FIELD	TYPE	DESCRIPTION
manifestBlobUrl	string	The blob URL for manifest file for inventory run.

## Inventory output

Each inventory rule generates a set of files in the specified inventory destination container for that rule. The inventory output is generated under the following path:

`https://<accountName>.blob.core.windows.net/<inventory-destination-container>/YYYY/MM/DD/HH-MM-SS/<ruleName>`

where:

- *accountName* is your Azure Blob Storage account name.
- *inventory-destination-container* is the destination container you specified in the inventory rule.
- *YYYY/MM/DD/HH-MM-SS* is the time when the inventory began to run.
- *ruleName* is the inventory rule name.

### Inventory files

Each inventory run for a rule generates the following files:

- **Inventory file:** An inventory run for a rule generates one or more CSV or Apache Parquet formatted files. If the matched object count is large, then multiple files are generated instead of a single file. Each such file contains matched objects and their metadata.

#### NOTE

Reports in the Apache Parquet format present dates in the following format:

`timestamp_millis [number of milliseconds since 1970-01-01 00:00:00 UTC]`.

For a CSV formatted file, the first row is always the schema row. The following image shows an inventory CSV file opened in Microsoft Excel.

A	B	C	D	E	F	G	H	I	J	K	
Name	Creation-Time	Last-Modified	Content-Length	Content-MD5	BlobType	AccessTier	AccessTierChangeTime	Snapshot	VersionId	IsCurrentVersion	
1 inventory-source-container/inventory-blob.txt	2020-10-06T11:2020-10-06T11:2		49180	fN0cmBldq2V+cj	BlockBlob	Cool	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z
2 inventory-source-container/inventory-blob.txt	2020-10-06T11:2020-10-06T11:2		49180	fN0cmBldq2V+cj	BlockBlob	Cool	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z
3 inventory-source-container/inventory-blob.txt	2020-10-06T11:2020-10-06T11:2		49180	fN0cmBldq2V+cj	BlockBlob	Cool	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z
4 inventory-source-container/inventory-blob.txt	2020-10-06T11:2020-10-06T11:2		49180	fN0cmBldq2V+cj	BlockBlob	Cool	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z
5 inventory-source-container/inventory-blob.txt	2020-10-06T11:2020-10-06T11:2		49180	fN0cmBldq2V+cj	BlockBlob	Cool	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z
6 inventory-source-container/inventory-blob.txt	2020-10-06T11:2020-10-06T11:2		49180	fN0cmBldq2V+cj	BlockBlob	Cool	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z	2020-10-06T11:28:12Z
7											

#### IMPORTANT

The blob paths that appear in an inventory file might not appear in any particular order.

- **Checksum file:** A checksum file contains the MD5 checksum of the contents of manifest.json file. The name of the checksum file is `<ruleName>-manifest.checksum`. Generation of the checksum file marks the completion of an inventory rule run.
- **Manifest file:** A manifest.json file contains the details of the inventory file(s) generated for that rule. The name of the file is `<ruleName>-manifest.json`. This file also captures the rule definition provided by the user and the path to the inventory for that rule. The following json shows the contents of a sample manifest.json file.

```
{
 "destinationContainer" : "inventory-destination-container",
 "endpoint" : "https://testaccount.blob.core.windows.net",
 "files" : [
 {
 "blob" : "2021/05/26/13-25-36/Rule_1/Rule_1.csv",
 "size" : 12710092
 }
],
 "inventoryCompletionTime" : "2021-05-26T13:35:56Z",
 "inventoryStartTime" : "2021-05-26T13:25:36Z",
 "ruleDefinition" : {
 "filters" : {
 "blobTypes" : ["blockBlob"],
 "includeBlobVersions" : false,
 "includeSnapshots" : false,
 "prefixMatch" : ["penner-test-container-100003"]
 },
 "format" : "csv",
 "objectType" : "blob",
 "schedule" : "daily",
 "schemaFields" : [
 "Name",
 "Creation-Time",
 "BlobType",
 "Content-Length",
 "LastAccessTime",
 "Last-Modified",
 "Metadata",
 "AccessTier"
]
 },
 "ruleName" : "Rule_1",
 "status" : "Succeeded",
 "summary" : {
 "objectCount" : 110000,
 "totalObjectSize" : 23789775
 },
 "version" : "1.0"
}
```

## Pricing and billing

Pricing for inventory is based on the number of blobs and containers that are scanned during the billing period. As an example, suppose an account contains one million blobs, and blob inventory is set to run once per week. After four weeks, four million blob entries will have been scanned.

Billing for blob inventory begins on October 1, 2021. Regional pricing will be published at that time. The baseline price without regional adjustment is approximately \$0.0025 USD per million entries scanned for blob storage and \$0.0035 USD if Data Lake Storage Gen2 is enabled. After inventory files are created, additional standard data storage and operations charges will be incurred for storing, reading, and writing the inventory-generated files in the account.

After an inventory report is complete, additional standard data storage and operations charges are incurred for storing, reading, and writing the inventory report in the storage account.

If a rule contains a prefix that overlaps with a prefix of any other rule, then the same blob can appear in more than one inventory report. In this case, you are billed for both instances. For example, assume that the `prefixMatch` element of one rule is set to `["inventory-blob-1", "inventory-blob-2"]`, and the `prefixMatch` element of another rule is set to `["inventory-blob-10", "inventory-blob-20"]`. An object named `inventory-blob-200` appears in both inventory reports.

Snapshots and versions of a blob also count towards billing even if you've set `includeSnapshots` and `includeVersions` filters to `false`. Those filter values don't affect billing. You can use them only to filter what appears in the report.

For more information about pricing for Azure Storage blob inventory, see [Azure Blob Storage pricing](#).

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## Known issues

This section describes limitations and known issues of the Azure Storage blob inventory feature.

### **Inventory job fails to complete for hierarchical namespace enabled accounts**

The inventory job might not complete within 2 days for an account with hundreds of millions of blobs and hierarchical namespace enabled. If this happens, no inventory file is created. If a job does not complete successfully, check subsequent jobs to see if they complete before contacting support. The performance of a job can vary, so if a job doesn't complete, it's possible that subsequent jobs will.

### **Inventory job cannot write inventory reports**

An object replication policy can prevent an inventory job from writing inventory reports to the destination container. Some other scenarios can archive the reports or make them immutable when they are partially completed. This can lead to inventory job failure.

## Next steps

- [Enable Azure Storage blob inventory reports](#)
- [Calculate the count and total size of blobs per container](#)
- [Manage the Azure Blob Storage lifecycle](#)

# Manage and find Azure Blob data with blob index tags

8/22/2022 • 15 minutes to read • [Edit Online](#)

As datasets get larger, finding a specific object in a sea of data can be difficult. Blob index tags provide data management and discovery capabilities by using key-value index tag attributes. You can categorize and find objects within a single container or across all containers in your storage account. As data requirements change, objects can be dynamically categorized by updating their index tags. Objects can remain in-place with their current container organization.

Blob index tags let you:

- Dynamically categorize your blobs using key-value index tags
- Quickly find specific tagged blobs across an entire storage account
- Specify conditional behaviors for blob APIs based on the evaluation of index tags
- Use index tags for advanced controls on features like [blob lifecycle management](#)

Consider a scenario where you have millions of blobs in your storage account, accessed by many different applications. You want to find all related data from a single project. You aren't sure what's in scope as the data can be spread across multiple containers with different naming conventions. However, your applications upload all data with tags based on their project. Instead of searching through millions of blobs and comparing names and properties, you can use `Project = Contoso` as your discovery criteria. Blob index will filter all containers across your entire storage account to quickly find and return just the set of 50 blobs from `Project = Contoso`.

To get started with examples on how to use blob index, see [Use blob index tags to manage and find data](#).

## Blob index tags and data management

Container and blob name prefixes are one-dimensional categorizations. Blob index tags allow for multi-dimensional categorization for [blob data types \(Block, Append, or Page\)](#). Multi-dimensional categorization is natively indexed by Azure Blob Storage so you can quickly find your data.

Consider the following five blobs in your storage account:

- `container1/transaction.csv`
- `container2/campaign.docx`
- `photos/bannerphoto.png`
- `archives/completed/2019review.pdf`
- `logs/2020/01/01/logfile.txt`

These blobs are separated using a prefix of `container/virtual folder/blob name`. You can set an index tag attribute of `Project = Contoso` on these five blobs to categorize them together while maintaining their current prefix organization. Adding index tags eliminates the need to move data by exposing the ability to filter and find data using the index.

## Setting blob index tags

Blob index tags are key-value attributes that can be applied to new or existing objects within your storage account. You can specify index tags during the upload process using [Put Blob](#), [Put Block List](#), or [Copy Blob](#) operations and the optional `x-ms-tags` header. If you already have blobs in your storage account, call [Set Blob Tags](#) passing a formatted XML document with the index tags in the body of the request.

#### IMPORTANT

Setting blob index tags can be performed by the [Storage Blob Data Owner](#) and by anyone with a Shared Access Signature that has permission to access the blob's tags (the `t` SAS permission).

In addition, RBAC users with the `Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write` permission can perform this operation.

You can apply a single tag on your blob to describe when your data was finished processing.

```
"processedDate" = '2020-01-01'
```

You can apply multiple tags on your blob to be more descriptive of the data.

```
"Project" = 'Contoso' "Classified" = 'True' "Status" = 'Unprocessed' "Priority" = '01'
```

To modify the existing index tag attributes, retrieve the existing tag attributes, modify the tag attributes, and replace with the [Set Blob Tags](#) operation. To remove all index tags from the blob, call the [Set Blob Tags](#) operation with no tag attributes specified. As blob index tags are a subresource to the blob data contents, [Set Blob Tags](#) doesn't modify any underlying content and doesn't change the blob's last-modified-time or eTag. You can create or modify index tags for all current base blobs. Index tags are also preserved for previous versions but they aren't passed to the blob index engine, so you cannot query index tags to retrieve previous versions. Tags on snapshots or soft deleted blobs cannot be modified.

The following limits apply to blob index tags:

- Each blob can have up to 10 blob index tags
- Tag keys must be between one and 128 characters
- Tag values must be between zero and 256 characters
- Tag keys and values are case-sensitive
- Tag keys and values only support string data types. Any numbers, dates, times, or special characters are saved as strings
- Tag keys and values must adhere to the following naming rules:
  - Alphanumeric characters:
    - a through z (lowercase letters)
    - A through Z (uppercase letters)
    - 0 through 9 (numbers)
  - Valid special characters: space, plus, minus, period, colon, equals, underscore, forward slash (`+-.:=/_`)

## Getting and listing blob index tags

Blob index tags are stored as a subresource alongside the blob data and can be retrieved independently from

the underlying blob data content. Blob index tags for a single blob can be retrieved with the [Get Blob Tags](#) operation. The [List Blobs](#) operation with the `include:tags` parameter will also return all blobs within a container along with their blob index tags.

#### IMPORTANT

Getting and listing blob index tags can be performed by the [Storage Blob Data Owner](#) and by anyone with a Shared Access Signature that has permission to access the blob's tags (the `t` SAS permission).

In addition, RBAC users with the `Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/read` permission can perform this operation.

For any blobs with at least one blob index tag, the `x-ms-tag-count` is returned in the [List Blobs](#), [Get Blob](#), and [Get Blob Properties](#) operations indicating the count of index tags on the blob.

## Finding data using blob index tags

The indexing engine exposes your key-value attributes into a multi-dimensional index. After you set your index tags, they exist on the blob and can be retrieved immediately. It may take some time before the blob index updates. After the blob index updates, you can use the native query and discovery capabilities offered by Blob Storage.

The [Find Blobs by Tags](#) operation enables you to get a filtered set of blobs whose index tags match a given query expression. [Find Blobs by Tags](#) supports filtering across all containers within your storage account or you can scope the filtering to just a single container. Since all the index tag keys and values are strings, relational operators use a lexicographic sorting.

#### IMPORTANT

Finding data using blob index tags can be performed by the [Storage Blob Data Owner](#) and by anyone with a Shared Access Signature that has permission to find blobs by tags (the `f` SAS permission).

In addition, RBAC users with the

`Microsoft.Storage/storageAccounts/blobServices/containers/blobs/filter/action` permission can perform this operation.

The following criteria applies to blob index filtering:

- Tag keys should be enclosed in double quotes ("")
- Tag values and container names should be enclosed in single quotes ('')
- The @ character is only allowed for filtering on a specific container name (for example, `@container = 'ContainerName'`)
- Filters are applied with lexicographic sorting on strings
- Same sided range operations on the same key are invalid (for example, `"Rank" > '10' AND "Rank" >= '15'`)
- When using REST to create a filter expression, characters should be URI encoded
- Tag queries are optimized for equality match using a single tag (e.g. `StoreID = "100"`). Range queries using a single tag involving `>`, `>=`, `<`, `<=` are also efficient. Any query using AND with more than one tag will not be as efficient. For example, `Cost > "01" AND Cost <= "100"` is efficient. `Cost > "01 AND StoreID = "2"` is not as efficient.

The below table shows all the valid operators for `Find Blobs by Tags`:

OPERATOR	DESCRIPTION	EXAMPLE
=	Equal	<code>"Status" = 'In Progress'</code>
>	Greater than	<code>"Date" &gt; '2018-06-18'</code>
>=	Greater than or equal	<code>"Priority" &gt;= '5'</code>
<	Less than	<code>"Age" &lt; '32'</code>
<=	Less than or equal	<code>"Priority" &lt;= '5'</code>
AND	Logical and	<code>"Rank" &gt;= '010' AND "Rank" &lt; '100'</code>
@container	Scope to a specific container	<code>@container = 'videofiles' AND "status" = 'done'</code>

#### NOTE

Be familiar with lexicographical ordering when setting and querying on tags.

- Numbers are sorted before letters. Numbers are sorted based on the first digit.
- Uppercase letters are sorted before lowercase letters.
- Symbols aren't standard. Some symbols are sorted before numeric values. Other symbols are sorted before or after letters.

## Conditional blob operations with blob index tags

In REST versions 2019-10-10 and higher, most [blob service APIs](#) now support a conditional header, `x-ms-if-tags`, such that the operation will only succeed if the specified blob index condition is met. If the condition isn't met, you'll get `error 412: The condition specified using HTTP conditional header(s) is not met`.

The `x-ms-if-tags` header may be combined with the other existing HTTP conditional headers (If-Match, If-None-Match, and so on). If multiple conditional headers are provided in a request, they all must evaluate true for the operation to succeed. All conditional headers are effectively combined with logical AND.

The below table shows the valid operators for conditional operations:

OPERATOR	DESCRIPTION	EXAMPLE
=	Equal	<code>"Status" = 'In Progress'</code>
<>	Not equal	<code>"Status" &lt;&gt; 'Done'</code>
>	Greater than	<code>"Date" &gt; '2018-06-18'</code>
>=	Greater than or equal	<code>"Priority" &gt;= '5'</code>
<	Less than	<code>"Age" &lt; '32'</code>

OPERATOR	DESCRIPTION	EXAMPLE
<=	Less than or equal	"Priority" <= '5'
AND	Logical and	"Rank" >= '010' AND "Rank" < '100'
OR	Logical or	"Status" = 'Done' OR "Priority" >= '05'

#### NOTE

There are two additional operators, not equal and logical or, that are allowed in the conditional `x-ms-if-tags` header for blob operations but do not exist in the `Find Blobs by Tags` operation.

## Platform integrations with blob index tags

Blob index tags not only help you categorize, manage, and search on your blob data, but also provide integration with other Blob Storage features, such as [lifecycle management](#).

### Lifecycle management

Using the `blobIndexMatch` as a rule filter in lifecycle management, you can move data to cooler tiers or delete data based on the index tags applied to your blobs. You can be more granular in your rules and only move or delete blobs if they match the specified tags criteria.

You can set a blob index match as a standalone filter set in a lifecycle rule to apply actions on tagged data. Or you can combine both a prefix and a blob index to match more specific data sets. Specifying multiple filters in a lifecycle rule applies a logical AND operation. The action will only apply if *all* filter criteria match.

The following sample lifecycle management rule applies to block blobs in a container called `videofiles`. The rule moves blobs to archive storage only if the data matches the blob index tag criteria of

```
"Status" == 'Processed' AND "Source" == 'RAW'.
```

- [Portal](#)
- [JSON](#)

**Action set**

Rule name	ArchiveProcessedSourceVideos
Status	Enabled
Blobs	Move to archive storage 1 days after blob last modification.
Snapshots	None

**Filter set**

Prefix match	videofiles
Blob Index match	Status == Processed Source == RAW

## Permissions and authorization

You can authorize access to blob index tags using one of the following approaches:

- Using Azure role-based access control (Azure RBAC) to grant permissions to an Azure Active Directory (Azure AD) security principal. Use Azure AD for superior security and ease of use. For more information

about using Azure AD with blob operations, see [Authorize access to data in Azure Storage](#).

- Using a shared access signature (SAS) to delegate access to blob index. For more information about shared access signatures, see [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#).
- Using the account access keys to authorize operations with Shared Key. For more information, see [Authorize with Shared Key](#).

Blob index tags are a subresource to the blob data. A user with permissions or a SAS token to read or write blobs may not have access to the blob index tags.

## Role-based access control

Callers using an [Azure AD identity](#) may be granted the following permissions to operate on blob index tags.

BLOB INDEX TAG OPERATIONS	AZURE RBAC ACTION
Set Blob Tags	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write
Get Blob Tags	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/read
Find Blobs by Tags	Microsoft.Storage/storageAccounts/blobServices/containers/blobs/filter/action

Additional permissions, separate from the underlying blob data, are required for index tag operations. The [Storage Blob Data Owner](#) role is granted permissions for all three blob index tag operations.

## SAS permissions

Callers using a [shared access signature \(SAS\)](#) may be granted scoped permissions to operate on blob index tags.

### Service SAS for a blob

The following permissions may be granted in a service SAS for a blob to allow access to blob index tags. The blob read (`r`) and write (`w`) permissions alone aren't enough to allow reading or writing its index tags.

PERMISSION	URI SYMBOL	ALLOWED OPERATIONS
Index tags	t	Get and set index tags for a blob

### Service SAS for a container

The following permissions may be granted in a service SAS for a container to allow filtering on blob tags. The blob list (`i`) permission isn't enough to allow filtering blobs by their index tags.

PERMISSION	URI SYMBOL	ALLOWED OPERATIONS
Index tags	f	Find blobs with index tags

### Account SAS

The following permissions may be granted in an account SAS to allow access to blob index tags and filtering on blob tags.

PERMISSION	URI SYMBOL	ALLOWED OPERATIONS
Index tags	t	Get and set index tags for a blob

PERMISSION	URI SYMBOL	ALLOWED OPERATIONS
Index tags	f	Find blobs with index tags

The blob read (`r`) and write (`w`) permissions alone aren't enough to allow reading or writing its index tags, and the list (`i`) permission isn't enough to allow filtering blobs by their index tags.

## Choosing between metadata and blob index tags

Both blob index tags and metadata provide the ability to store arbitrary user-defined key-value properties alongside a blob resource. Both can be retrieved and set directly, without returning or altering the contents of the blob. It's possible to use both metadata and index tags.

Only index tags are automatically indexed and made searchable by the native Blob Storage service. Metadata can't be natively indexed or searched. You must use a separate service such as [Azure Search](#). Blob index tags have additional permissions for reading, filtering, and writing that are separate from the underlying blob data. Metadata uses the same permissions as the blob and is returned as HTTP headers by the [Get Blob](#) and [Get Blob Properties](#) operations. Blob index tags are encrypted at rest using a [Microsoft-managed key](#). Metadata is encrypted at rest using the same encryption key specified for blob data.

The following table summarizes the differences between metadata and blob index tags:

	METADATA	BLOB INDEX TAGS
Limits	No numerical limit, 8 KB total, case insensitive	10 tags per blob max, 768 bytes per tag, case sensitive
Updates	Not allowed on archive tier, <code>Set Blob Metadata</code> replaces all existing metadata, <code>Set Blob Metadata</code> changes the blob's last-modified-time	Allowed for all access tiers, <code>Set Blob Tags</code> replaces all existing tags, <code>Set Blob Tags</code> doesn't change the blob's last-modified-time
Storage	Stored with the blob data	Subresource of the blob data
Indexing & Querying	Must use a separate service such as Azure Search	Indexing and querying capabilities built into Blob Storage
Encryption	Encrypted at rest with the same encryption key used for blob data	Encrypted at rest with a Microsoft-managed encryption key
Pricing	Size of metadata is included in the storage costs for a blob	Fixed cost per index tag
Header response	Metadata returned as headers in <code>Get Blob</code> and <code>Get Blob Properties</code>	Tag count returned by <code>Get Blob</code> or <code>Get Blob Properties</code> , tags returned only by <code>Get Blob Tags</code> and <code>List Blobs</code>
Permissions	Read or write permissions to blob data extends to metadata	Additional permissions are required to read, filter, or write index tags
Naming	Metadata names must adhere to the naming rules for C# identifiers	Blob index tags support a wider range of alphanumeric characters

## Pricing

You're charged for the monthly average number of index tags within a storage account. There's no cost for the indexing engine. Requests to Set Blob Tags, Get Blob Tags, and Find Blob Tags are charged at the current respective transaction rates. Note that the number of list transactions consumed when doing a Find Blobs by Tag transaction is equal to the number of clauses in the request. For example, the query (StoreID = 100) is one list transaction. The query (StoreID = 100 AND SKU = 10010) is two list transactions. See [Block Blob pricing to learn more](#).

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## Conditions and known issues

This section describes known issues and conditions.

- Only general-purpose v2 accounts are supported. Premium block blob, legacy blob, and accounts with a hierarchical namespace enabled aren't supported. General-purpose v1 accounts won't be supported.
- Uploading page blobs with index tags doesn't persist the tags. Set the tags after uploading a page blob.
- If Blob storage versioning is enabled, you can still use index tags on the current version. Index tags are preserved for previous versions, but those tags aren't passed to the blob index engine, so you cannot use them to retrieve previous versions. If you promote a previous version to the current version, then the tags of that previous version become the tags of the current version. Because those tags are associated with the current version, they are passed to the blob index engine and you can query them.
- There is no API to determine if index tags are indexed.
- Lifecycle management only supports equality checks with blob index match.
- `Copy Blob` doesn't copy blob index tags from the source blob to the new destination blob. You can specify the tags you want applied to the destination blob during the copy operation.

## FAQ

### Can blob index help me filter and query content inside my blobs?

No, if you need to search within your blob data, use query acceleration or Azure search.

### Are there any requirements on index tag values?

Blob index tags only support string data types and querying returns results with lexicographical ordering. For numbers, zero pad the number. For dates and times, store as an ISO 8601 compliant format.

### Are blob index tags and Azure Resource Manager tags related?

No, Resource Manager tags help organize control plane resources such as subscriptions, resource groups, and storage accounts. Index tags provide blob management and discovery on the data plane.

## Next steps

For an example of how to use blob index, see [Use blob index to manage and find data](#).

Learn about [lifecycle management](#) and set a rule with blob index matching.

# Search over Azure Blob Storage content

8/22/2022 • 7 minutes to read • [Edit Online](#)

Searching across the variety of content types stored in Azure Blob Storage can be a difficult problem to solve, but [Azure Cognitive Search](#) provides deep integration at the content layer, extracting and inferring textual information, which can then be queried in a search index.

In this article, review the basic workflow for extracting content and metadata from blobs and sending it to a [search index](#) in Azure Cognitive Search. The resulting index can be queried using full text search. Optionally, you can send processed blob content to a [knowledge store](#) for non-search scenarios.

## NOTE

Already familiar with the workflow and composition? [Configure a blob indexer](#) is your next step.

## What it means to add full text search to blob data

Azure Cognitive Search is a standalone search service that supports indexing and query workloads over user-defined indexes that contain your remote searchable content hosted in the cloud. Co-locating your searchable content with the query engine is necessary for performance, returning results at a speed users have come to expect from search queries.

Cognitive Search integrates with Azure Blob Storage at the indexing layer, importing your blob content as search documents that are indexed into *inverted indexes* and other query structures that support free-form text queries and filter expressions. Because your blob content is indexed into a search index, you can use the full range of query features in Azure Cognitive Search to find information in your blob content.

Inputs are your blobs, in a single container, in Azure Blob Storage. Blobs can be almost any kind of text data. If your blobs contain images, you can add [AI enrichment](#) to create and extract text from images.

Output is always an Azure Cognitive Search index, used for fast text search, retrieval, and exploration in client applications. In between is the indexing pipeline architecture itself. The pipeline is based on the *indexer* feature, discussed further on in this article.

Once the index is created and populated, it exists independently of your blob container, but you can re-run indexing operations to refresh your index based on changed documents. Timestamp information on individual blobs is used for change detection. You can opt for either scheduled execution or on-demand indexing as the refresh mechanism.

## Resources used in a blob-search solution

You need Azure Cognitive Search, Azure Blob Storage, and a client. Cognitive Search is typically one of several components in a solution, where your application code issues query API requests and handles the response. You might also write application code to handle indexing, although for proof-of-concept testing and impromptu tasks, it's common to use the Azure portal as the search client.

Within Blob Storage, you'll need a container that provides source content. You can set file inclusion and exclusion criteria, and specify which parts of a blob are indexed in Cognitive Search.

You can start directly in your Storage Account portal page.

1. In the left navigation page under **Data management**, select **Azure search** to select or create a search

service.

- Follow the steps in the wizard to extract and optionally create searchable content from your blobs. The workflow is the [Import data wizard](#).

The screenshot shows the Azure portal interface for managing blob storage. On the left, a sidebar lists categories such as Data management, Configuration, and Monitoring. The main area is titled 'demo-blob-storage | Azure search' and contains a configuration form. The 'Data source' dropdown is set to 'Azure Blob Storage'. The 'Data source name' field is empty. The 'Data to extract' dropdown is set to 'Content and metadata'. The 'Parsing mode' dropdown is set to 'Default'. The 'Connection string' dropdown shows a placeholder 'DefaultEndpointsProtocol=https;AccountName=blo...'. Under 'Managed identity authentication', the 'None' radio button is selected. The 'Container name' field is empty, and the 'Blob folder' field contains 'your/folder/here'. A 'Description' field is labeled '(optional)'. At the bottom, buttons for 'Previous: Select a search service' and 'Next: Add cognitive skills (Optional)' are visible.

- Use [Search explorer](#) in the search portal page to query your content.

The wizard is the best place to start, but you'll discover more flexible options when you [configure a blob indexer](#) yourself. You can call the REST APIs using a tool like Postman or Visual Studio Code. [Tutorial: Index and search semi-structured data \(JSON blobs\) in Azure Cognitive Search](#) walks you through the steps of calling the REST API in Postman.

## How blobs are indexed

By default, most blobs are indexed as a single search document in the index, including blobs with structured content, such as JSON or CSV, which are indexed as a single chunk of text. However, for JSON or CSV documents that have an internal structure (delimiters), you can assign parsing modes to generate individual search documents for each line or element:

- [Indexing JSON blobs](#)
- [Indexing CSV blobs](#)

A compound or embedded document (such as a ZIP archive, a Word document with embedded Outlook email containing attachments, or an .MSG file with attachments) is also indexed as a single document. For example, all images extracted from the attachments of an .MSG file will be returned in the normalized\_images field. If you have images, consider adding [AI enrichment](#) to get more search utility from that content.

Textual content of a document is extracted into a string field named "content". You can also extract standard and user-defined metadata.

## NOTE

Azure Cognitive Search imposes [indexer limits](#) on how much text it extracts depending on the pricing tier. A warning will appear in the indexer status response if documents are truncated.

# Use a Blob indexer for content extraction

An *indexer* is a data-source-aware subservice in Cognitive Search, equipped with internal logic for sampling data, reading metadata data, retrieving data, and serializing data from native formats into JSON documents for subsequent import.

Blobs in Azure Storage are indexed using the [blob indexer](#). You can invoke this indexer by using the **Azure search** command in Azure Storage, the **Import data** wizard, a REST API, or the .NET SDK. In code, you use this indexer by setting the type, and by providing connection information that includes an Azure Storage account along with a blob container. You can subset your blobs by creating a virtual directory, which you can then pass as a parameter, or by filtering on a file type extension.

An indexer "[cracks a document](#)", opening a blob to inspect content. After connecting to the data source, it's the first step in the pipeline. For blob data, this is where PDF, Office docs, and other content types are detected. Document cracking with text extraction is no charge. If your blobs contain image content, images are ignored unless you [add AI enrichment](#). Standard indexing applies only to text content.

The Blob indexer comes with configuration parameters and supports change tracking if the underlying data provides sufficient information. You can learn more about the core functionality in [Blob indexer](#).

## Supported access tiers

Blob storage [access tiers](#) include hot, cool, and archive. Only hot and cool can be accessed by indexers.

## Supported content types

By running a Blob indexer over a container, you can extract text and metadata from the following content types with a single query:

- CSV (see [Indexing CSV blobs](#))
- EML
- EPUB
- GZ
- HTML
- JSON (see [Indexing JSON blobs](#))
- KML (XML for geographic representations)
- Microsoft Office formats: DOCX/DOC/DOCM, XLSX/XLS/XLSM, PPTX/PPT/PPTM, MSG (Outlook emails), XML (both 2003 and 2006 WORD XML)
- Open Document formats: ODT, ODS, ODP
- PDF
- Plain text files (see also [Indexing plain text](#))
- RTF
- XML
- ZIP

## Controlling which blobs are indexed

You can control which blobs are indexed, and which are skipped, by the blob's file type or by setting properties on the blob themselves, causing the indexer to skip over them.

Include specific file extensions by setting `"indexedFileNameExtensions"` to a comma-separated list of file extensions (with a leading dot). Exclude specific file extensions by setting `"excludedFileNameExtensions"` to the extensions that should be skipped. If the same extension is in both lists, it will be excluded from indexing.

```
PUT /indexers/[indexer name]?api-version=2020-06-30
{
 "parameters" : {
 "configuration" : {
 "indexedFileNameExtensions" : ".pdf, .docx",
 "excludedFileNameExtensions" : ".png, .jpeg"
 }
 }
}
```

## Add "skip" metadata the blob

The indexer configuration parameters apply to all blobs in the container or folder. Sometimes, you want to control how *individual blobs* are indexed.

Add the following metadata properties and values to blobs in Blob Storage. When the indexer encounters this property, it will skip the blob or its content in the indexing run.

PROPERTY NAME	PROPERTY VALUE	EXPLANATION
<code>"AzureSearch_Skip"</code>	<code>"true"</code>	Instructs the blob indexer to completely skip the blob. Neither metadata nor content extraction is attempted. This is useful when a particular blob fails repeatedly and interrupts the indexing process.
<code>"AzureSearch_SkipContent"</code>	<code>"true"</code>	This is equivalent to the <code>"dataToExtract" : "allMetadata"</code> setting described <a href="#">above</a> scoped to a particular blob.

## Indexing blob metadata

A common scenario that makes it easy to sort through blobs of any content type is to [index both custom metadata and system properties](#) for each blob. In this way, information for all blobs is indexed regardless of document type, stored in an index in your search service. Using your new index, you can then proceed to sort, filter, and facet across all Blob storage content.

### NOTE

Blob Index tags are natively indexed by the Blob storage service and exposed for querying. If your blobs' key/value attributes require indexing and filtering capabilities, Blob Index tags should be leveraged instead of metadata.

To learn more about Blob Index, see [Manage and find data on Azure Blob Storage with Blob Index](#).

## Search blob content in a search index

The output of an indexer is a search index, used for interactive exploration using free text and filtered queries in a client app. For initial exploration and verification of content, we recommend starting with [Search Explorer](#) in the portal to examine document structure. In Search explorer, you can use:

- [Simple query syntax](#)
- [Full query syntax](#)

- [Filter expression syntax](#)

A more permanent solution is to gather query inputs and present the response as search results in a client application. The following C# tutorial explains how to build a search application: [Create your first application in Azure Cognitive Search](#).

## Next steps

- [Upload, download, and list blobs with the Azure portal \(Azure Blob storage\)](#)
- [Set up a blob indexer \(Azure Cognitive Search\)](#)

# Azure Storage migration overview

8/22/2022 • 6 minutes to read • [Edit Online](#)

This article focuses on storage migrations to Azure and provides guidance on the following storage migration scenarios:

- Migration of unstructured data, such as files and objects
- Migration of block-based devices, such as disks and storage area networks (SANs)

## Migration of unstructured data

Migration of unstructured data includes following scenarios:

- File migration from network attached storage (NAS) to one of the Azure file offerings:
  - [Azure Files](#)
  - [Azure NetApp Files](#)
  - [independent software vendor \(ISV\) solutions.](#)
- Object migration from object storage solutions to the Azure object storage platform:
  - [Azure Blob Storage](#)
  - [Azure Data Lake Storage.](#)

## Migration phases

A full migration consists of several different phases: discovery, assessment, and migration.

DISCOVERY	ASSESSMENT	MIGRATION
- Discover sources to be migrated	- Assess applicable target service - Technical vs. cost considerations	- Initial migration - Resync - Final switch over

### Discovery phase

In the discovery phase, you determine all sources that need to be migrated like SMB shares, NFS exports, or object namespaces. You can do this phase manually, or use automated tools.

### Assessment phase

The assessment phase is critical in understanding available options for the migration. To reduce the risk during migration, and to avoid common pitfalls follow these three steps:

ASSESSMENT PHASE STEPS	OPTIONS
<b>Choose a target storage service</b>	- Azure Blob Storage and Data Lake Storage - Azure Files - Azure NetApp Files - ISV solutions
<b>Select a migration method</b>	- Online - Offline - Combination of both
<b>Choose the best migration tool for the job</b>	- Commercial tools (Azure and ISV) - Open source

There are several commercial (ISV) tools that can help with the assessment phase. See the [comparison matrix](#).

#### Choose a target storage service

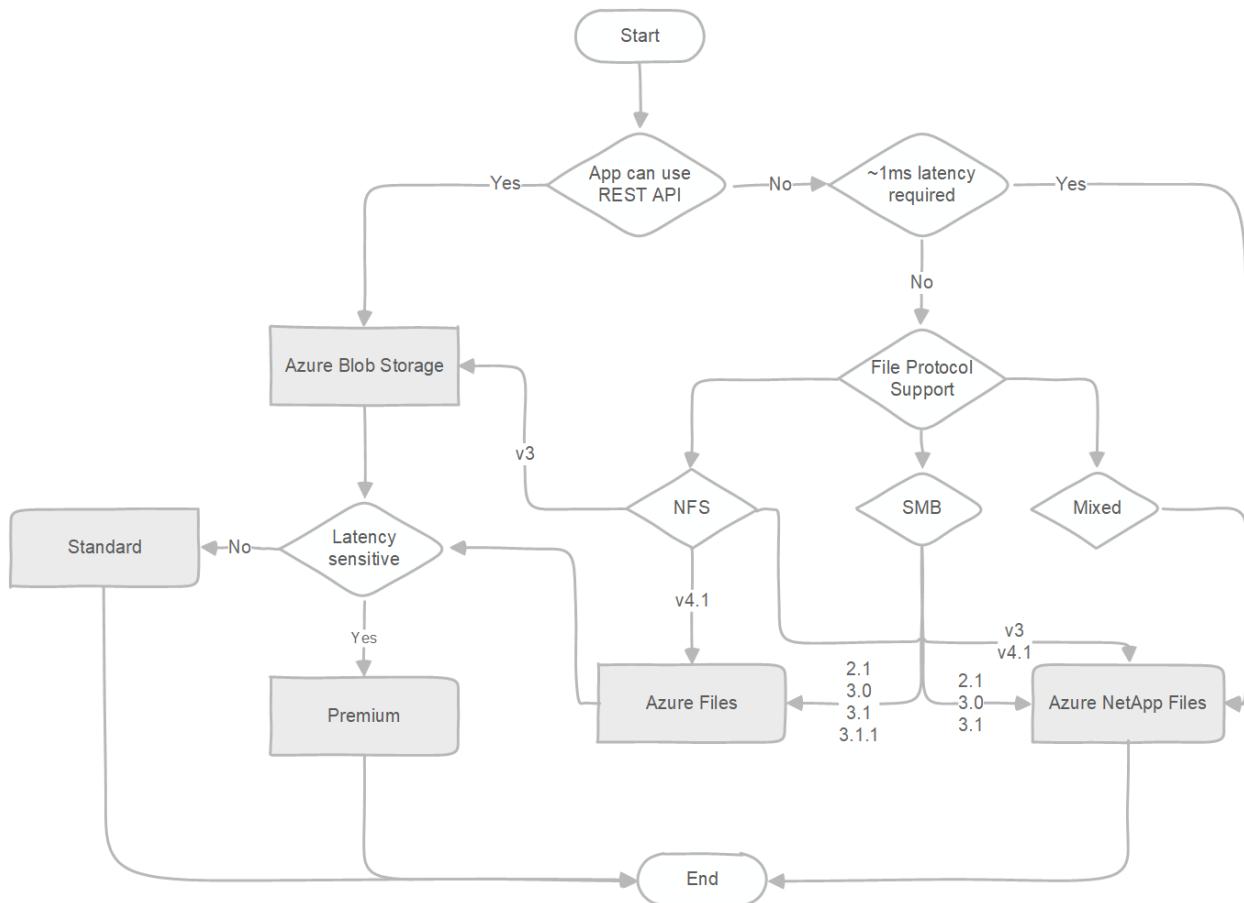
Choosing a target storage service depends on the application or users who access the data. The correct choice depends on both technical and financial aspects. First, do a technical assessment to assess possible targets and determine which services satisfy the requirements. Next, do a financial assessment to determine the best choice.

To help select the target storage service for the migration, evaluate the following aspects of each service:

- Protocol support
- Performance characteristics
- Limits of the target storage service

The following diagram is a simplified decision tree that helps guide you to the recommended Azure file service. If native Azure services do not satisfy requirements, a variety of [independent software vendor \(ISV\) solutions](#) will.

After you finish the technical assessment, and select the proper target, do a cost assessment to determine the most cost-effective option.



To keep the decision tree simple, limits of the target storage service aren't incorporated in the diagram. To find out more about current limits, and to determine whether you need to modify your choices based on them, see:

- [Storage account limits](#)
- [Blob Storage limits](#)
- [Azure Files scalability and performance targets](#)
- [Azure NetApp Files resource limits](#)

If any of the limits pose a blocker for using a service, Azure supports several storage vendors that offer their solutions on Azure Marketplace. For information about validated ISV partners that provide file services, see [Azure Storage partners for primary and secondary storage](#).

#### Select the migration method

There are two basic migration methods for storage migrations.

- **Online.** The online method uses the network for data migration. Either the public internet or [Azure ExpressRoute](#) can be used. If the service doesn't have a public endpoint, you must use a VPN with public internet.
- **Offline.** The offline method uses one of the [Azure Data Box](#) devices.

The decision to use an online method versus an offline method depends on the available network bandwidth. The online method is preferred in cases where there's sufficient network bandwidth to perform a migration within the needed timeline.

It's possible to use a combination of both methods, offline method for the initial bulk migration and an online method for incremental migration of changes. Using both methods simultaneously requires a high level of coordination and isn't recommended for this reason. If you choose to use both methods isolate the data sets that are migrated online from the data sets that are migrated offline.

For more information about the different migration methods and guidelines, see [Choose an Azure solution for data transfer](#) and [Migrate to Azure file shares](#).

#### Choose the best migration tool for the job

There are various migration tools that you can use to perform the migration. Some are open source like AzCopy, robocopy, xcopy, and rsync while others are commercial. List of available commercial tools and comparison between them is available on our [comparison matrix](#).

Open-source tools are well suited for small-scale migrations. For migration from Windows file servers to Azure Files, Microsoft recommends starting with Azure Files native capability and using [Azure File Sync](#). For more complex migrations consisting of different sources, large capacity, or special requirements like throttling or detailed reporting with audit capabilities, commercial tools are the best choice. These tools make the migration easier and reduce the risk significantly. Most commercial tools can also perform the discovery, which provides a valuable input for the assessment.

#### **Migration phase**

The migration phase is the final migration step that does data movement and migration. Typically, you'll run through the migration phase several times to accomplish an easier switchover. The migration phase consists of the following steps:

1. **Initial migration.** The initial migration step migrates all the data from the source to the target. This step migrates the bulk of the data that needs to be migrated.
2. **Resync.** A resync operation migrates any data that was changed after the initial migration step. You can repeat this step several times if there are numerous changes. The goal of running multiple resync operations is to reduce the time it takes for the final step. For inactive data and for data that has no changes (like backup or archive data), you can skip this step.
3. **Final switchover.** The final switchover step switches the active usage of the data from the source to the target and retires the source.

The duration of the migration for unstructured data depends on several aspects. Outside of the chosen method, the most critical factors are the total size of the data and file size distribution. The bigger the total data set, the longer the migration time. The smaller the average file size, the longer the migration time. If you have a large number of small files consider archiving them in larger files (like to a .tar or .zip file), if applicable, to reduce the total migration time.

## Migration of block-based devices

Migration of block-based devices is typically done as part of virtual machine or physical host migration. It's a common misconception to delay block storage decisions until after the migration. Making these decisions ahead of time with appropriate considerations for workload requirements leads to a smoother migration to the cloud.

To explore workloads to migrate and approach to take, see the [Azure Disk Storage documentation](#), and resources on the [Disk Storage product page](#). You can learn about which disks fit your requirements, and the latest capabilities such as [disk bursting](#). Migration of block based devices can be done in two ways:

- For migration of full virtual machines together with the underlying block-based devices, see the [Azure Migrate](#) documentation
- For migration of block based devices only, and more complexed use cases, use [Cirrus Migrate Cloud](#).

## See also

- [Choose an Azure solution for data transfer](#)
- [Commercial migration tools comparison](#)
- [Migrate to Azure file shares](#)
- [Migrate to Data Lake Storage with WANdisco LiveData Platform for Azure](#)
- [Copy or move data to Azure Storage with AzCopy](#)
- [Migrate large datasets to Azure Blob Storage with AzReplicate](#)

# Choose an Azure solution for data transfer

8/22/2022 • 3 minutes to read • [Edit Online](#)

This article provides an overview of some of the common Azure data transfer solutions. The article also links out to recommended options depending on the network bandwidth in your environment and the size of the data you intend to transfer.

## Types of data movement

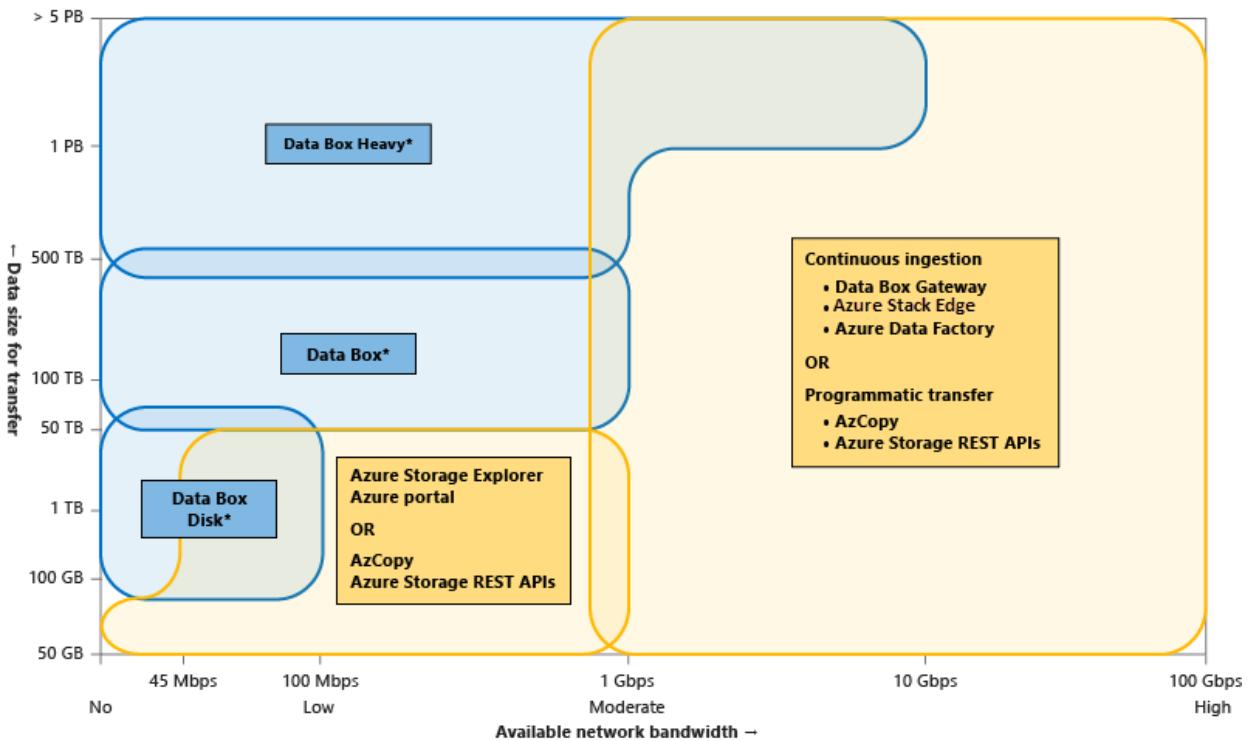
Data transfer can be offline or over the network connection. Choose your solution depending on your:

- **Data size** - Size of the data intended for transfer,
- **Transfer frequency** - One-time or periodic data ingestion, and
- **Network** – Bandwidth available for data transfer in your environment.

The data movement can be of the following types:

- **Offline transfer using shippable devices** - Use physical shippable devices when you want to do offline one-time bulk data transfer. Microsoft sends you a disk, or a secure specialized device. Alternatively, you can purchase and ship your own disks. You copy data to the device and then ship it to Azure where the data is uploaded. The available options for this case are Data Box Disk, Data Box, Data Box Heavy, and Import/Export (use your own disks).
- **Network Transfer** - You transfer your data to Azure over your network connection. This can be done in many ways.
  - **Graphical interface** - If you occasionally transfer just a few files and do not need to automate the data transfer, you can choose a graphical interface tool such as Azure Storage Explorer or a web-based exploration tool in Azure portal.
  - **Scripted or programmatic transfer** - You can use optimized software tools that we provide or call our REST APIs/SDKs directly. The available scriptable tools are AzCopy, Azure PowerShell, and Azure CLI. For programmatic interface, use one of the SDKs for .NET, Java, Python, Node/JS, C++, Go, PHP or Ruby.
  - **On-premises devices** - We supply you a physical or virtual device that resides in your datacenter and optimizes data transfer over the network. These devices also provide a local cache of frequently used files. The physical device is the Azure Stack Edge and the virtual device is the Data Box Gateway. Both run permanently in your premises and connect to Azure over the network.
  - **Managed data pipeline** - You can set up a cloud pipeline to regularly transfer files between several Azure services, on-premises or a combination of two. Use Azure Data Factory to set up and manage data pipelines, and move and transform data for analysis.

The following visual illustrates the guidelines to choose the various Azure data transfer tools depending upon the network bandwidth available for transfer, data size intended for transfer, and frequency of the transfer.



\* The upper limits of the offline transfer devices - Data Box Disk, Data Box, and Data Box Heavy can be extended by placing multiple orders of a device type.

## Selecting a data transfer solution

Answer the following questions to help select a data transfer solution:

- Is your available network bandwidth limited or non-existent, and you want to transfer large datasets?

If yes, see: [Scenario 1: Transfer large datasets with no or low network bandwidth](#).

- Do you want to transfer large datasets over network and you have a moderate to high network bandwidth?

If yes, see: [Scenario 2: Transfer large datasets with moderate to high network bandwidth](#).

- Do you want to occasionally transfer just a few files over the network?

If yes, see [Scenario 3: Transfer small datasets with limited to moderate network bandwidth](#).

- Are you looking for point-in-time data transfer at regular intervals?

If yes, use the scripted/programmatic options outlined in [Scenario 4: Periodic data transfers](#).

- Are you looking for on-going, continuous data transfer?

If yes, use the options in [Scenario 4: Periodic data transfers](#).

## Data transfer feature in Azure portal

You can also go to your Azure Storage account in Azure portal and select the **Data transfer** feature. Provide the network bandwidth in your environment, the size of the data you want to transfer, and the frequency of data transfer. You will see the optimum data transfer solutions corresponding to the information that you have provided.

## Next steps

- Get an introduction to Azure Storage Explorer.
- Read an overview of AzCopy.
- Quickstart: Upload, download, and list blobs with PowerShell
- Quickstart: Create, download, and list blobs with Azure CLI
- Learn about:
  - Azure Data Box, Azure Data Box Disk, and Azure Data Box Heavy for offline transfers.
  - Azure Data Box Gateway and Azure Stack Edge for online transfers.
- Learn what is Azure Data Factory.
- Use the REST APIs to transfer data
  - In .NET
  - In Java

# Data transfer for large datasets with low or no network bandwidth

8/22/2022 • 2 minutes to read • [Edit Online](#)

This article provides an overview of the data transfer solutions when you have limited to no network bandwidth in your environment and you are planning to transfer large data sets. The article also describes the recommended data transfer options and the respective key capability matrix for this scenario.

To understand an overview of all the available data transfer options, go to [Choose an Azure data transfer solution](#).

## Offline transfer or network transfer

Large datasets imply that you have few TBs to few PBs of data. You have limited to no network bandwidth, your network is slow, or it is unreliable. Also:

- You are limited by costs of network transfer from your Internet Service Providers (ISPs).
- Security or organizational policies do not allow outbound connections when dealing with sensitive data.

In all the above instances, use a physical device to do a one-time bulk data transfer. Choose from Data Box Disk, Data Box, Data Box Heavy devices which are supplied by Microsoft, or Import/Export using your own disks.

To confirm whether a physical device is the right option, use the following table. It shows the projected time for network data transfer, for various available bandwidths (assuming 90% utilization). If network transfer is projected to be too slow, you should use a physical device.

Data size ↓ Network bandwidth →	45 Mbps (T3)	100 Mbps	1 Gbps	10 Gbps
<b>1 TB</b>	2 days	1 day	3 hours	15 minutes
<b>10 TB</b>	<b>23 days</b>	11 days	1 day	3 hours
<b>35 TB</b>	<b>82 days</b>	<b>37 days</b>	4 days	9 hours
<b>80 TB</b>	<b>187 days</b>	<b>84 days</b>	8 days	20 hours
<b>100 TB</b>	<b>234 days</b>	<b>105 days</b>	11 days	1 day
<b>200 TB</b>	<b>1 year</b>	<b>211 days</b>	<b>21 days</b>	2 days
<b>500 TB</b>	<b>3 years</b>	<b>1 year</b>	<b>53 days</b>	5 days
<b>1 PB</b>	<b>7 years</b>	<b>3 years</b>	<b>108 days</b>	11 days
<b>2 PB</b>	<b>13 years</b>	<b>6 years</b>	<b>216 days</b>	22 days
<b>5 PB</b>	<b>33 years</b>	<b>15 years</b>	<b>1 year</b>	<b>54 days</b>

Key:
Use a Data Box Disk instead
Use a Data Box instead
Use a Data Box Heavy instead
Use the network

## Recommended options

The options available in this scenario are devices for Azure Data Box offline transfer or Azure Import/Export.

- **Azure Data Box family for offline transfers** – Use devices from Microsoft-supplied Data Box devices to move large amounts of data to Azure when you're limited by time, network availability, or costs. Copy on-premises data using tools such as Robocopy. Depending on the data size intended for transfer, you can choose from Data Box Disk, Data Box, or Data Box Heavy.
- **Azure Import/Export** – Use Azure Import/Export service by shipping your own disk drives to securely import large amounts of data to Azure Blob storage and Azure Files. This service can also be used to transfer data from Azure Blob storage to disk drives and ship to your on-premises sites.

# Comparison of key capabilities

The following table summarizes the differences in key capabilities.

	DATA BOX DISK	DATA BOX	DATA BOX HEAVY	IMPORT/EXPORT
<b>Data size</b>	Up to 35 TBs	Up to 80 TBs per device	Up to 800 TB per device	Variable
<b>Data type</b>	Azure Blobs	Azure Blobs Azure Files	Azure Blobs Azure Files	Azure Blobs Azure Files
<b>Form factor</b>	5 SSDs per order	1 X 50-lbs. desktop-sized device per order	1 X ~500-lbs. large device per order	Up to 10 HDDs/SSDs per order
<b>Initial setup time</b>	Low (15 mins)	Low to moderate (<30 mins)	Moderate (1-2 hours)	Moderate to difficult (variable)
<b>Send data to Azure</b>	Yes	Yes	Yes	Yes
<b>Export data from Azure</b>	No	No	No	Yes
<b>Encryption</b>	AES 128-bit	AES 256-bit	AES 256-bit	AES 128-bit
<b>Hardware</b>	Microsoft supplied	Microsoft supplied	Microsoft supplied	Customer supplied
<b>Network interface</b>	USB 3.1/SATA	RJ 45, SFP+	RJ45, QSFP+	SATA II/SATA III
<b>Partner integration</b>	Some	High	High	Some
<b>Shipping</b>	Microsoft managed	Microsoft managed	Microsoft managed	Customer managed
<b>Use when data moves</b>	Within a commerce boundary	Within a commerce boundary	Within a commerce boundary	Across geographic boundaries, e.g. US to EU
<b>Pricing</b>	Pricing	Pricing	Pricing	Pricing

## Next steps

- Understand how to
  - [Transfer data with Data Box Disk](#).
  - [Transfer data with Data Box](#).
  - [Transfer data with Import/Export](#).

# Data transfer for large datasets with moderate to high network bandwidth

8/22/2022 • 5 minutes to read • [Edit Online](#)

This article provides an overview of the data transfer solutions when you have moderate to high network bandwidth in your environment and you are planning to transfer large datasets. The article also describes the recommended data transfer options and the respective key capability matrix for this scenario.

To understand an overview of all the available data transfer options, go to [Choose an Azure data transfer solution](#).

## Scenario description

Large datasets refer to data sizes in the order of TBs to PBs. Moderate to high network bandwidth refers to 100 Mbps to 10 Gbps.

## Recommended options

The options recommended in this scenario depend on whether you have moderate network bandwidth or high network bandwidth.

### Moderate network bandwidth (100 Mbps - 1 Gbps)

With moderate network bandwidth, you need to project the time for data transfer over the network.

Use the following table to estimate the time and based on that, choose between an offline transfer or over the network transfer. The table shows the projected time for network data transfer, for various available network bandwidths (assuming 90% utilization).

Data size ↓ Network bandwidth →	45 Mbps (T3)	100 Mbps	1 Gbps	10 Gbps
<b>1 TB</b>	2 days	1 day	3 hours	15 minutes
<b>10 TB</b>	23 days	11 days	1 day	3 hours
<b>35 TB</b>	82 days	37 days	4 days	9 hours
<b>80 TB</b>	187 days	84 days	8 days	20 hours
<b>100 TB</b>	234 days	105 days	11 days	1 day
<b>200 TB</b>	1 year	211 days	21 days	2 days
<b>500 TB</b>	3 years	1 year	53 days	5 days
<b>1 PB</b>	7 years	3 years	108 days	11 days
<b>2 PB</b>	13 years	6 years	216 days	22 days
<b>5 PB</b>	33 years	15 years	1 year	54 days

Key:
Use a Data Box Disk instead
Use a Data Box instead
Use a Data Box Heavy instead
Use the network

- If the network transfer is projected to be too slow, you should use a physical device. The recommended options in this case are the offline transfer devices from Azure Data Box family or Azure Import/Export using your own disks.
  - **Azure Data Box family for offline transfers** – Use devices from Microsoft-supplied Data Box devices to move large amounts of data to Azure when you're limited by time, network availability, or costs. Copy on-premises data using tools such as Robocopy. Depending on the data size intended for transfer, you can choose from Data Box Disk, Data Box, or Data Box Heavy.
  - **Azure Import/Export** – Use Azure Import/Export service by shipping your own disk drives to

securely import large amounts of data to Azure Blob storage and Azure Files. This service can also be used to transfer data from Azure Blob storage to disk drives and ship to your on-premises sites.

- If the network transfer is projected to be reasonable, then you can use any of the following tools detailed in [High network bandwidth](#).

### High network bandwidth (1 Gbps - 100 Gbps)

If the available network bandwidth is high, use one of the following tools.

- **AzCopy** - Use this command-line tool to easily copy data to and from Azure Blobs, Files, and Table storage with optimal performance. AzCopy supports concurrency and parallelism, and the ability to resume copy operations when interrupted.
- **Azure Storage REST APIs/SDKs** – When building an application, you can develop the application against Azure Storage REST APIs and use the Azure SDKs offered in multiple languages.
- **Azure Data Box family for online transfers** – Azure Stack Edge and Data Box Gateway are online network devices that can move data into and out of Azure. Use Azure Stack Edge physical device when there is a simultaneous need for continuous ingestion and pre-processing of the data prior to upload. Data Box Gateway is a virtual version of the device with the same data transfer capabilities. In each case, the data transfer is managed by the device.
- **Azure Data Factory** – Data Factory should be used to scale out a transfer operation, and if there is a need for orchestration and enterprise grade monitoring capabilities. Use Data Factory to regularly transfer files between several Azure services, on-premises, or a combination of the two. with Data Factory, you can create and schedule data-driven workflows (called pipelines) that ingest data from disparate data stores and automate data movement and data transformation.

## Comparison of key capabilities

The following tables summarize the differences in key capabilities for the recommended options.

### Moderate network bandwidth

If using offline data transfer, use the following table to understand the differences in key capabilities.

	DATA BOX DISK	DATA BOX	DATA BOX HEAVY	IMPORT/EXPORT
Data size	Up to 35 TBs	Up to 80 TBs per device	Up to 800 TB per device	Variable
Data type	Azure Blobs	Azure Blobs Azure Files	Azure Blobs Azure Files	Azure Blobs Azure Files
Form factor	5 SSDs per order	1 X 50-lbs. desktop-sized device per order	1 X ~500-lbs. large device per order	Up to 10 HDDs/SSDs per order
Initial setup time	Low (15 mins)	Low to moderate (<30 mins)	Moderate (1-2 hours)	Moderate to difficult (variable)
Send data to Azure	Yes	Yes	Yes	Yes
Export data from Azure	No	No	No	Yes
Encryption	AES 128-bit	AES 256-bit	AES 256-bit	AES 128-bit

	DATA BOX DISK	DATA BOX	DATA BOX HEAVY	IMPORT/EXPORT
<b>Hardware</b>	Microsoft supplied	Microsoft supplied	Microsoft supplied	Customer supplied
<b>Network interface</b>	USB 3.1/SATA	RJ 45, SFP+	RJ45, QSFP+	SATA II/SATA III
<b>Partner integration</b>	Some	High	High	Some
<b>Shipping</b>	Microsoft managed	Microsoft managed	Microsoft managed	Customer managed
<b>Use when data moves</b>	Within a commerce boundary	Within a commerce boundary	Within a commerce boundary	Across geographic boundaries, e.g. US to EU
<b>Pricing</b>	<a href="#">Pricing</a>	<a href="#">Pricing</a>	<a href="#">Pricing</a>	<a href="#">Pricing</a>

If using online data transfer, use the table in the following section for high network bandwidth.

### High network bandwidth

	TOOLS AZCOPY, AZURE POWERSHELL, AZURE CLI	AZURE STORAGE REST APIs, SDKS	DATA BOX GATEWAY OR AZURE STACK EDGE	AZURE DATA FACTORY
<b>Data type</b>	Azure Blobs, Azure Files, Azure Tables	Azure Blobs, Azure Files, Azure Tables	Azure Blobs, Azure Files	Supports 70+ data connectors for data stores and formats
<b>Form factor</b>	Command-line tools	Programmatic interface	Microsoft supplies a virtual or physical device	Service in Azure portal
<b>Initial one-time setup</b>	Easy	Moderate	Easy (<30 minutes) to moderate (1-2 hours)	Extensive
<b>Data pre-processing</b>	No	No	Yes (With Edge compute)	Yes
<b>Transfer from other clouds</b>	No	No	No	Yes
<b>User type</b>	IT Pro or dev	Dev	IT Pro	IT Pro
<b>Pricing</b>	Free, data egress charges apply	Free, data egress charges apply	<a href="#">Azure Stack Edge pricing</a> <a href="#">Data Box Gateway pricing</a>	<a href="#">Pricing</a>

## Next steps

- [Learn how to transfer data with Import/Export.](#)
- Understand how to:
  - [Transfer data with Data Box Disk.](#)

- Transfer data with Data Box.
- Transfer data with AzCopy.
- Transfer data with Data Box Gateway.
- Transform data with Azure Stack Edge before sending to Azure.
- Learn how to transfer data with Azure Data Factory.
- Use the REST APIs to transfer data:
  - In .NET
  - In Java

# Data transfer for small datasets with low to moderate network bandwidth

8/22/2022 • 2 minutes to read • [Edit Online](#)

This article provides an overview of the data transfer solutions when you have low to moderate network bandwidth in your environment and you are planning to transfer small datasets. The article also describes the recommended data transfer options and the respective key capability matrix for this scenario.

To understand an overview of all the available data transfer options, go to [Choose an Azure data transfer solution](#).

## Scenario description

Small datasets refer to data sizes in the order of GBs to a few TBs. Low to moderate network bandwidth implies 45 Mbps (T3 connection in datacenter) to 1 Gbps.

- If you are transferring only a handful of files and you don't need to automate data transfer, consider the tools with a graphical interface.
- If you are comfortable with system administration, consider command line or programmatic/scripting tools.

## Recommended options

The options recommended in this scenario are:

- **Graphical interface tools** such as Azure Storage Explorer and Azure Storage in Azure portal. These provide an easy way to view your data and quickly transfer a few files.
  - **Azure Storage Explorer** - This cross-platform tool lets you manage the contents of your Azure storage accounts. It allows you to upload, download, and manage blobs, files, queues, tables, and Azure Cosmos DB entities. Use it with Blob storage to manage blobs and folders, as well as upload and download blobs between your local file system and Blob storage, or between storage accounts.
  - **Azure portal** - Azure Storage in Azure portal provides a web-based interface to explore files and upload new files one at a time. This is a good option if you do not want to install any tools or issue commands to quickly explore your files, or to simply upload a handful of new ones.
- **Scripting/programmatic tools** such as AzCopy/PowerShell/Azure CLI and Azure Storage REST APIs.
  - **AzCopy** - Use this command-line tool to easily copy data to and from Azure Blobs, Files, and Table storage with optimal performance. AzCopy supports concurrency and parallelism, and the ability to resume copy operations when interrupted.
  - **Azure PowerShell** - For users comfortable with system administration, use the Azure Storage module in Azure PowerShell to transfer data.
  - **Azure CLI** - Use this cross-platform tool to manage Azure services and upload data to Azure Storage.
  - **Azure Storage REST APIs/SDKs** – When building an application, you can develop the application against Azure Storage REST APIs/SDKs and use the Azure client libraries offered in multiple languages.

## Comparison of key capabilities

The following table summarizes the differences in key capabilities.

FEATURE	AZURE STORAGE EXPLORER	AZURE PORTAL	AZCOPY AZURE POWERSHELL AZURE CLI	AZURE STORAGE REST APIs OR SDKS
Availability	Download and install Standalone tool	Web-based exploration tools in Azure portal	Command line tool	Programmable interfaces in .NET, Java, Python, JavaScript, C++, Go, Ruby and PHP
Graphical interface	Yes	Yes	No	No
Supported platforms	Windows, Mac, Linux	Web-based	Windows, Mac, Linux	All platforms
Allowed Blob storage operations for blobs and folders	Upload Download Manage	Upload Download Manage	Upload Download Manage	Yes, customizable
Allowed Data Lake Gen1 storage operations for files and folders	Upload Download Manage	No	Upload Download Manage	No
Allowed File storage operations for files and directories	Upload Download Manage	Upload Download Manage	Upload Download Manage	Yes, customizable
Allowed Table storage operations for tables	Manage	No	Table support in AzCopy v7	Yes, customizable
Allowed Queue storage	Manage	No	No	Yes, is customizable

## Next steps

- Learn how to [transfer data with Azure Storage Explorer](#).
- [Transfer data with AzCopy](#)

# Solutions for periodic data transfer

8/22/2022 • 3 minutes to read • [Edit Online](#)

This article provides an overview of the data transfer solutions when you are transferring data periodically. Periodic data transfer over the network can be categorized as recurring at regular intervals or continuous data movement. The article also describes the recommended data transfer options and the respective key capability matrix for this scenario.

To understand an overview of all the available data transfer options, go to [Choose an Azure data transfer solution](#).

## Recommended options

The recommended options for periodic data transfer fall into two categories depending on whether the transfer is recurring or continuous.

- **Scripted/programmatic tools** – For data transfer that occurs at regular intervals, use the scripted and programmatic tools such as AzCopy and Azure Storage REST APIs. These tools are targeted towards IT professionals and developers.
  - **AzCopy** - Use this command-line tool to easily copy data to and from Azure Blobs, Files, and Table storage with optimal performance. AzCopy supports concurrency and parallelism, and the ability to resume copy operations when interrupted.
  - **Azure Storage REST APIs/SDKs** – When building an application, you can develop the application against Azure Storage REST APIs and use the Azure SDKs offered in multiple languages. The REST APIs can also leverage the Azure Storage Data Movement Library designed especially for the high-performance copying of data to and from Azure.
- **Continuous data ingestion tools** – For continuous, ongoing data ingestion, you can select one of the following options.
  - **Object replication** - Object replication asynchronously copies block blobs between containers in a source and destination storage account. Use object replication as a solution to keep containers in two different storage accounts in sync.
  - **Azure Data Factory** – Data Factory should be used to scale out a transfer operation, and if there is a need for orchestration and enterprise grade monitoring capabilities. Use Azure Data Factory to set up a cloud pipeline that regularly transfers files between several Azure services, on-premises, or a combination of the two. Azure Data Factory lets you orchestrate data-driven workflows that ingest data from disparate data stores and automate data movement and data transformation.
  - **Azure Data Box family for online transfers** - Data Box Edge and Data Box Gateway are online network devices that can move data into and out of Azure. Data Box Edge uses artificial intelligence (AI)-enabled Edge compute to pre-process data before upload. Data Box Gateway is a virtual version of the device with the same data transfer capabilities.

Data Box online transfer device or Azure Data Factory are set up by IT professionals and can transparently automate data transfer.

## Comparison of key capabilities

The following table summarizes the differences in key capabilities.

### Scripted/Programmatic network data transfer

CAPABILITY	AZCOPY	AZURE STORAGE REST APIs
Form factor	Command-line tool from Microsoft	Customers develop against Storage REST APIs using Azure client libraries
Initial one-time setup	Minimal	Moderate, variable development effort
Data Format	Azure Blobs, Azure Files, Azure Tables	Azure Blobs, Azure Files, Azure Tables
Performance	Already optimized	Optimize as you develop
Pricing	Free, data egress charges apply	Free, data egress charges apply

### Continuous data ingestion over network

FEATURE	DATA BOX GATEWAY	DATA BOX EDGE	AZURE DATA FACTORY
Form factor	Virtual device	Physical device	Service in Azure portal, agent on-premises
Hardware	Your hypervisor	Supplied by Microsoft	NA
Initial setup effort	Low (<30 mins.)	Moderate (~couple hours)	Large (~days)
Data Format	Azure Blobs, Azure Files	Azure Blobs, Azure Files	Supports 70+ data connectors for data stores and formats
Data pre-processing	No	Yes, via Edge compute	Yes
Local cache (to store on-premises data)	Yes	Yes	No
Transfer from other clouds	No	No	Yes
Pricing	Pricing	Pricing	Pricing

## Next steps

- [Transfer data with AzCopy](#).
- [More information on data transfer with Storage REST APIs](#).
- Understand how to:
  - [Transfer data with Data Box Gateway](#).
  - [Transform data with Data Box Edge before sending to Azure](#).
- [Learn how to transfer data with Azure Data Factory](#).

# Monitoring Azure Blob Storage

8/22/2022 • 21 minutes to read • [Edit Online](#)

When you have critical applications and business processes that rely on Azure resources, you want to monitor those resources for their availability, performance, and operation. This article describes the monitoring data that's generated by Azure Blob Storage and how you can use the features of Azure Monitor to analyze alerts on this data.

## Monitor overview

The [Overview](#) page in the Azure portal for each Blob storage resource includes a brief view of the resource usage, such as requests and hourly billing. This information is useful, but only a small amount of the monitoring data is available. Some of this data is collected automatically and is available for analysis as soon as you create the resource. You can enable additional types of data collection with some configuration.

## What is Azure Monitor?

Azure Blob Storage creates monitoring data by using [Azure Monitor](#), which is a full stack monitoring service in Azure. Azure Monitor provides a complete set of features to monitor your Azure resources and resources in other clouds and on-premises.

Start with the article [Monitoring Azure resources with Azure Monitor](#) which describes the following:

- What is Azure Monitor?
- Costs associated with monitoring
- Monitoring data collected in Azure
- Configuring data collection
- Standard tools in Azure for analyzing and alerting on monitoring data

The following sections build on this article by describing the specific data gathered from Azure Storage. Examples show how to configure data collection and analyze this data with Azure tools.

## Monitoring data

Azure Blob Storage collects the same kinds of monitoring data as other Azure resources, which are described in [Monitoring data from Azure resources](#).

See [Azure Blob Storage monitoring data reference](#) for detailed information on the metrics and logs metrics created by Azure Blob Storage.

Metrics and logs in Azure Monitor support only Azure Resource Manager storage accounts. Azure Monitor doesn't support classic storage accounts. If you want to use metrics or logs on a classic storage account, you need to migrate to an Azure Resource Manager storage account. For more information, see [Migrate to Azure Resource Manager](#).

You can continue using classic metrics and logs if you want to. In fact, classic metrics and logs are available in parallel with metrics and logs in Azure Monitor. The support remains in place until Azure Storage ends the service on legacy metrics and logs.

## Collection and routing

Platform metrics and the Activity log are collected automatically, but can be routed to other locations by using a diagnostic setting.

To collect resource logs, you must create a diagnostic setting. When you create the setting, choose **blob** as the type of storage that you want to enable logs for. Then, specify one of the following categories of operations for which you want to collect logs.

CATEGORY	DESCRIPTION
StorageRead	Read operations on objects.
StorageWrite	Write operations on objects.
StorageDelete	Delete operations on objects.

**NOTE**

Data Lake Storage Gen2 doesn't appear as a storage type. That's because Data Lake Storage Gen2 is a set of capabilities available to Blob storage.

## Creating a diagnostic setting

This section shows you how to create a diagnostic setting by using the Azure portal, PowerShell, and the Azure CLI. This section provides steps specific to Azure Storage. For general guidance about how to create a diagnostic setting, see [Create diagnostic setting to collect platform logs and metrics in Azure](#).

**TIP**

You can also create a diagnostic setting by using an Azure Resource manager template or by using a policy definition. A policy definition can ensure that a diagnostic setting is created for every account that is created or updated.

This section doesn't describe templates or policy definitions.

- To view an Azure Resource Manager template that creates a diagnostic setting, see [Diagnostic setting for Azure Storage](#).
- To learn how to create a diagnostic setting by using a policy definition, see [Azure Policy built-in definitions for Azure Storage](#).

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

1. Sign in to the Azure portal.
2. Navigate to your storage account.
3. In the **Monitoring** section, click **Diagnostic settings**.

The screenshot shows the Azure portal interface with the title bar "Microsoft Azure". The left sidebar has sections for Monitoring (with Insights, Alerts, Metrics, Workbooks, and Diagnostic settings), Logs, Monitoring (classic), Metrics (classic), Diagnostic settings (classic), and Usage (classic). The "Diagnostic settings" link is highlighted with a red box. The main content area shows a search bar and filter options for Subscription (contoso), Resource group (contoso-resource-group), Resource type (Storage accounts), and Resource (contoso). Below this, a table lists resources with their names, resource types, resource groups, and diagnostics status (all set to "Disabled").

Name	Resource type	Resource group	Diagnostics status
contoso	Storage account	contoso-resource-group	Disabled
blob	Storage account	contoso-resource-group	Disabled
queue	Storage account	contoso-resource-group	Disabled
table	Storage account	contoso-resource-group	Disabled
file	Storage account	contoso-resource-group	Disabled

4. Choose **blob** as the type of storage that you want to enable logs for.
5. Click **Add diagnostic setting**.

The **Diagnostic settings** page appears.

## Diagnostic setting



 Save  Discard  Delete  Feedback

A diagnostic setting specifies a list of categories of platform logs and/or metrics that you want to collect from a resource, and one or more destinations that you would stream them to. Normal usage charges for the destination will occur. [Learn more about the different log categories and contents of those logs](#)

Diagnostic setting name \*

### Logs

#### Categories

- StorageRead
- StorageWrite
- StorageDelete

#### Metrics

- Transaction

### Destination details

- Send to Log Analytics workspace
- Archive to a storage account
- Stream to an event hub
- Send to partner solution

6. In the **Name** field of the page, enter a name for this Resource log setting. Then, select which operations you want logged (read, write, and delete operations), and where you want the logs to be sent.

#### Archive logs to a storage account

If you choose to archive your logs to a storage account, you'll pay for the volume of logs that are sent to the storage account. For specific pricing, see the [Platform Logs](#) section of the [Azure Monitor pricing](#) page. You can't send logs to the same storage account that you are monitoring with this setting. This would lead to recursive logs in which a log entry describes the writing of another log entry. You must create an account or use another existing account to store log information.

1. Select the **Archive to a storage account** checkbox, and then select the **Configure** button.
2. In the **Storage account** drop-down list, select the storage account that you want to archive your logs to, and then select the **Save** button.

#### IMPORTANT

You can't set a retention policy. However, you can manage the retention policy of a log container by defining a lifecycle management policy. To learn how, see [Optimize costs by automating Azure Blob Storage access tiers](#).

#### NOTE

Before you choose a storage account as the export destination, see [Archive Azure resource logs](#) to understand prerequisites on the storage account.

#### Stream logs to Azure Event Hubs

If you choose to stream your logs to an event hub, you'll pay for the volume of logs that are sent to the event hub. For specific pricing, see the [Platform Logs](#) section of the [Azure Monitor pricing](#) page. You'll need access to an existing event hub, or you'll need to create one before you complete this step.

1. Select the **Stream to an event hub** checkbox, and then select the **Configure** button.
2. In the **Select an event hub** pane, choose the namespace, name, and policy name of the event hub that you want to stream your logs to.
3. Select the **Save** button.

#### Send logs to Azure Log Analytics

1. Select the **Send to Log Analytics** checkbox, select a log analytics workspace, and then select the **Save** button. You'll need access to an existing log analytics workspace, or you'll need to create one before you complete this step.

#### IMPORTANT

You can't set a retention policy. However, you can manage the data retention period of Log Analytics at the workspace level or even specify different retention settings by data type. To learn how, see [Change the data retention period](#).

#### Send to a partner solution

You can also send platform metrics and logs to certain Azure Monitor partners. You must first install a partner integration into your subscription. Configuration options will vary by partner. Check the [Azure Monitor partner integrations documentation](#) for details.

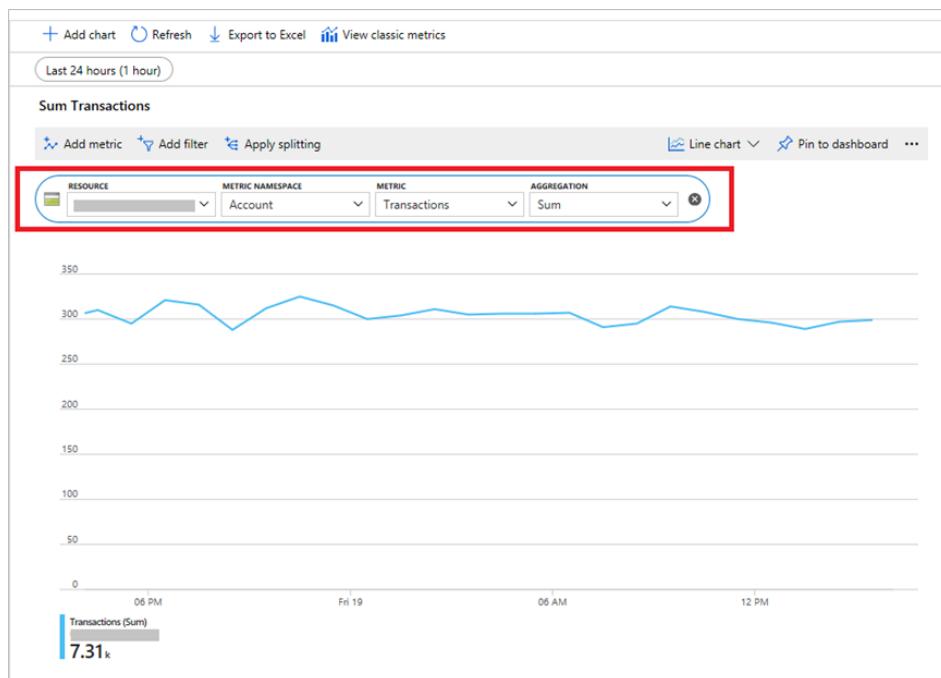
## Analyzing metrics

For a list of all Azure Monitor support metrics, which includes Azure Blob Storage, see [Azure Monitor supported metrics](#).

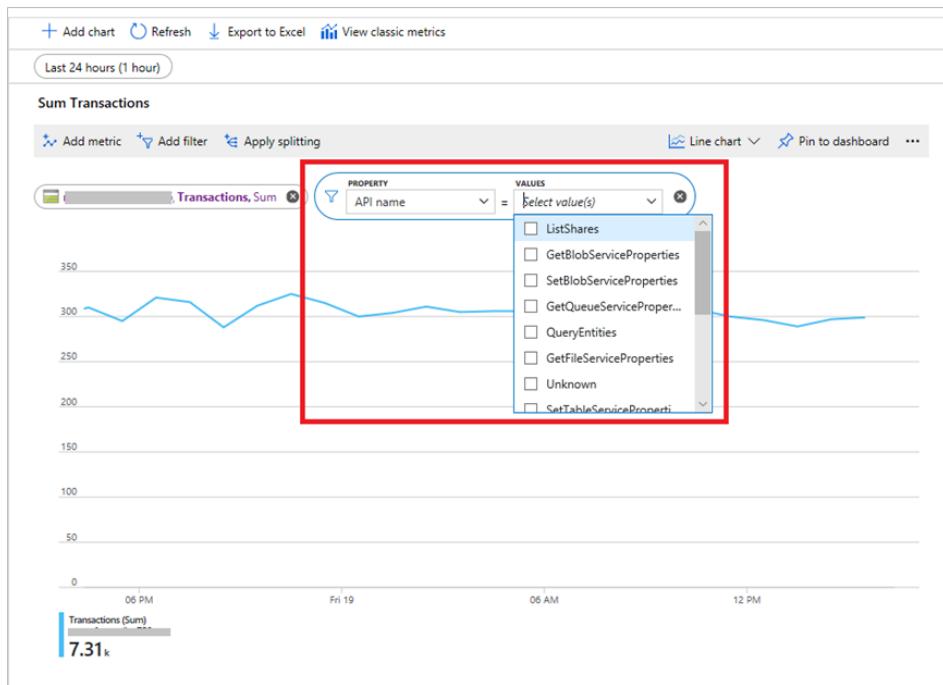
- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

You can analyze metrics for Azure Storage with metrics from other Azure services by using Metrics Explorer. Open Metrics Explorer by choosing **Metrics** from the **Azure Monitor** menu. For details on using this tool, see [Getting started with Azure Metrics Explorer](#).

This example shows how to view **Transactions** at the account level.



For metrics that support dimensions, you can filter the metric with the desired dimension value. This example shows how to view **Transactions** at the account level on a specific operation by selecting values for the **API Name** dimension.



For a complete list of the dimensions that Azure Storage supports, see [Metrics dimensions](#).

Metrics for Azure Blob Storage are in these namespaces:

- Microsoft.Storage/storageAccounts
- Microsoft.Storage/storageAccounts/blobServices

## Analyze metrics by using code

Azure Monitor provides the [.NET SDK](#) to read metric definition and values. The [sample code](#) shows how to use the SDK with different parameters. You need to use `0.18.0-preview` or a later version for storage metrics.

In these examples, replace the `<resource-ID>` placeholder with the resource ID of the entire storage account or the Blob storage service. You can find these resource IDs on the **Endpoints** pages of your storage account in the Azure portal.

Replace the `<subscription-ID>` variable with the ID of your subscription. For guidance on how to obtain values for `<tenant-ID>`, `<application-ID>`, and `<AccessKey>`, see [Use the portal to create an Azure AD application and service principal that can access resources](#).

### List the account-level metric definition

The following example shows how to list a metric definition at the account level:

```
public static async Task ListStorageMetricDefinition()
{
 var resourceId = "<resource-ID>";
 var subscriptionId = "<subscription-ID>";
 var tenantId = "<tenant-ID>";
 var applicationId = "<application-ID>";
 var accessKey = "<AccessKey>";

 MonitorManagementClient readOnlyClient = AuthenticateWithReadOnlyClient(tenantId, applicationId,
accessKey, subscriptionId).Result;
 IEnumerable<MetricDefinition> metricDefinitions = await
readOnlyClient.MetricDefinitions.ListAsync(resourceUri: resourceId, cancellationToken: new
CancellationToken());

 foreach (var metricDefinition in metricDefinitions)
 {
 // Enumerate metric definition:
 // Id
 // ResourceId
 // Name
 // Unit
 // MetricAvailabilities
 // PrimaryAggregationType
 // Dimensions
 // IsDimensionRequired
 }
}
```

### Reading account-level metric values

The following example shows how to read `UsedCapacity` data at the account level:

```

public static async Task ReadStorageMetricValue()
{
 var resourceId = "<resource-ID>";
 var subscriptionId = "<subscription-ID>";
 var tenantId = "<tenant-ID>";
 var applicationId = "<application-ID>";
 var accessKey = "<AccessKey>";

 MonitorClient readOnlyClient = AuthenticateWithReadOnlyClient(tenantId, applicationId, accessKey,
 subscriptionId).Result;

 Microsoft.Azure.Management.Monitor.Models.Response Response;

 string startDate = DateTime.Now.AddHours(-3).ToUniversalTime().ToString("o");
 string endDate = DateTime.Now.ToUniversalTime().ToString("o");
 string timeSpan = startDate + "/" + endDate;

 Response = await readOnlyClient.Metrics.ListAsync(
 resourceUri: resourceId,
 timespan: timeSpan,
 interval: System.TimeSpan.FromHours(1),
 metricnames: "UsedCapacity",

 aggregation: "Average",
 resultType: ResultType.Data,
 cancellationToken: CancellationToken.None);

 foreach (var metric in Response.Value)
 {
 // Enumerate metric value
 // Id
 // Name
 // Type
 // Unit
 // Timeseries
 // - Data
 // - Metadatavalues
 }
}

```

#### **Reading multidimensional metric values**

For multidimensional metrics, you need to define metadata filters if you want to read metric data on specific dimension values.

The following example shows how to read metric data on the metric supporting multidimension:

```

public static async Task ReadStorageMetricValueTest()
{
 // Resource ID for blob storage
 var resourceId =
"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccou
nts/{storageAccountName}/blobServices/default";
 var subscriptionId = "<subscription-ID>";
 // How to identify Tenant ID, Application ID and Access Key:
 https://azure.microsoft.com/documentation/articles/resource-group-create-service-principal-portal/
 var tenantId = "<tenant-ID>";
 var applicationId = "<application-ID>";
 var accessKey = "<AccessKey>";

 MonitorManagementClient readOnlyClient = AuthenticateWithReadOnlyClient(tenantId, applicationId,
accessKey, subscriptionId).Result;

 Microsoft.Azure.Management.Monitor.Models.Response Response;

 string startDate = DateTime.Now.AddHours(-3).ToUniversalTime().ToString("o");
 string endDate = DateTime.Now.ToUniversalTime().ToString("o");
 string timeSpan = startDate + "/" + endDate;
 // It's applicable to define meta data filter when a metric support dimension
 // More conditions can be added with the 'or' and 'and' operators, example: BlobType eq 'BlockBlob'
 or BlobType eq 'PageBlob'
 ODataQuery<MetadataValue> odataFilterMetrics = new ODataQuery<MetadataValue>(
 string.Format("BlobType eq '{0}'", "BlockBlob"));

 Response = readOnlyClient.Metrics.List(
 resourceUri: resourceId,
 timespan: timeSpan,
 interval: System.TimeSpan.FromHours(1),
 metricnames: "BlobCapacity",
 odataQuery: odataFilterMetrics,
 aggregation: "Average",
 resultType: ResultType.Data);

 foreach (var metric in Response.Value)
 {
 //Enumerate metric value
 // Id
 // Name
 // Type
 // Unit
 // Timeseries
 // - Data
 // - Metadatavalues
 }
}

```

## Analyzing logs

You can access resource logs either as a blob in a storage account, as event data, or through Log Analytics queries.

For a detailed reference of the fields that appear in these logs, see [Azure Blob Storage monitoring data reference](#).

Log entries are created only if there are requests made against the service endpoint. For example, if a storage account has activity in its blob endpoint but not in its table or queue endpoints, only logs that pertain to the blob service are created. Azure Storage logs contain detailed information about successful and failed requests to a storage service. This information can be used to monitor individual requests and to diagnose issues with a storage service. Requests are logged on a best-effort basis.

### Log authenticated requests

The following types of authenticated requests are logged:

- Successful requests
- Failed requests, including timeout, throttling, network, authorization, and other errors
- Requests that use a shared access signature (SAS) or OAuth, including failed and successful requests
- Requests to analytics data (classic log data in the **\$logs** container and class metric data in the **\$metric** tables)

Requests made by the Blob storage service itself, such as log creation or deletion, aren't logged. For a full list of the logged data, see [Storage logged operations and status messages](#) and [Storage log format](#).

### Log anonymous requests

The following types of anonymous requests are logged:

- Successful requests
- Server errors

- Timeout errors for both client and server
- Failed GET requests with the error code 304 (Not Modified)

All other failed anonymous requests aren't logged. For a full list of the logged data, see [Storage logged operations and status messages](#) and [Storage log format](#).

#### Accessing logs in a storage account

Logs appear as blobs stored to a container in the target storage account. Data is collected and stored inside a single blob as a line-delimited JSON payload. The name of the blob follows this naming convention:

```
https://<destination-storage-account>.blob.core.windows.net/insights-logs-<storage-operation>/resourceId=/subscriptions/<subscription-ID>/resourceGroups/<resource-group-name>/providers/Microsoft.Storage/storageAccounts/<source-storage-account>/blobServices/default/y=<year>/m=<month>/d=<day>/h=<hour>/m=<minute>/PT1H.json
```

Here's an example:

```
https://mylogstorageaccount.blob.core.windows.net/insights-logs-storagewrite/resourceId=/subscriptions/208841be-a4v3-4234-9450-08b90c09f4/resourceGroups/myresourcegroup/providers/Microsoft.Storage/storageAccounts/mystorageaccount/blobServices/default/y=2019/m=07/d=30/
```

#### Accessing logs in an event hub

Logs sent to an event hub aren't stored as a file, but you can verify that the event hub received the log information. In the Azure portal, go to your event hub and verify that the **incoming messages** count is greater than zero.



You can access and read log data that's sent to your event hub by using security information and event management and monitoring tools. For more information, see [What can I do with the monitoring data being sent to my event hub?](#).

#### Accessing logs in a Log Analytics workspace

You can access logs sent to a Log Analytics workspace by using Azure Monitor log queries.

For more information, see [Get started with Log Analytics in Azure Monitor](#).

Data is stored in the **StorageBlobLog** table. Logs for Data Lake Storage Gen2 do not appear in a dedicated table. That's because Data Lake Storage Gen2 is not a service. It's a set of capabilities that you can enable in your storage account. If you've enabled those capabilities, logs will continue to appear in the **StorageBlobLog** table.

#### Sample Kusto queries

Here are some queries that you can enter in the **Log search** bar to help you monitor your Blob storage. These queries work with the [new language](#).

#### IMPORTANT

When you select **Logs** from the storage account resource group menu, Log Analytics is opened with the query scope set to the current resource group. This means that log queries will only include data from that resource group. If you want to run a query that includes data from other resources or data from other Azure services, select **Logs** from the **Azure Monitor** menu. See [Log query scope and time range in Azure Monitor Log Analytics](#) for details.

Use these queries to help you monitor your Azure Storage accounts:

- To list the 10 most common errors over the last three days.

```
StorageBlobLogs
| where TimeGenerated > ago(3d) and StatusText !contains "Success"
| summarize count() by StatusText
| top 10 by count_desc
```

- To list the top 10 operations that caused the most errors over the last three days.

```
StorageBlobLogs
| where TimeGenerated > ago(3d) and StatusText !contains "Success"
| summarize count() by OperationName
| top 10 by count_desc
```

- To list the top 10 operations with the longest end-to-end latency over the last three days.

```
StorageBlobLogs
| where TimeGenerated > ago(3d)
| top 10 by DurationMs desc
| project TimeGenerated, OperationName, DurationMs, ServerLatencyMs, ClientLatencyMs = DurationMs -
ServerLatencyMs
```

- To list all operations that caused server-side throttling errors over the last three days.

```
StorageBlobLogs
| where TimeGenerated > ago(3d) and StatusText contains "ServerBusy"
| project TimeGenerated, OperationName, StatusCode, StatusText
```

- To list all requests with anonymous access over the last three days.

```
StorageBlobLogs
| where TimeGenerated > ago(3d) and AuthenticationType == "Anonymous"
| project TimeGenerated, OperationName, AuthenticationType, Uri
```

- To create a pie chart of operations used over the last three days.

```
StorageBlobLogs
| where TimeGenerated > ago(3d)
| summarize count() by OperationName
| sort by count_desc
| render piechart
```

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## FAQ

### Does Azure Storage support metrics for Managed Disks or Unmanaged Disks?

No. Azure Compute supports the metrics on disks. For more information, see [Per disk metrics for Managed and Unmanaged Disks](#).

## Next steps

Get started with any of these guides.

GUIDE	DESCRIPTION
<a href="#">Gather metrics from your Azure Blob Storage containers</a>	Create charts that show metrics (Contains step-by-step guidance).
<a href="#">Monitor, diagnose, and troubleshoot your Azure Storage</a>	Troubleshoot storage account issues (contains step-by-step guidance).
<a href="#">Monitor storage with Azure Monitor Storage insights</a>	A unified view of storage performance, capacity, and availability
<a href="#">Best practices for monitoring Azure Blob Storage</a>	Guidance for common monitoring and troubleshooting scenarios.

GUIDE	DESCRIPTION
<a href="#">Getting started with Azure Metrics Explorer</a>	A tour of Metrics Explorer.
<a href="#">Overview of Log Analytics in Azure Monitor</a>	A tour of Log Analytics.
<a href="#">Azure Monitor Metrics overview</a>	The basics of metrics and metric dimensions
<a href="#">Azure Monitor Logs overview</a>	The basics of logs and how to collect and analyze them
<a href="#">Transition to metrics in Azure Monitor</a>	Move from Storage Analytics metrics to metrics in Azure Monitor.
<a href="#">Azure Blob Storage monitoring data reference</a>	A reference of the logs and metrics created by Azure Blob Storage
<a href="#">Troubleshoot performance issues</a>	Common performance issues and guidance about how to troubleshoot them.
<a href="#">Troubleshoot availability issues</a>	Common availability issues and guidance about how to troubleshoot them.
<a href="#">Troubleshoot client application errors</a>	Common issues with connecting clients and how to troubleshoot them.

# Transition to metrics in Azure Monitor

8/22/2022 • 4 minutes to read • [Edit Online](#)

On August 31, 2023 Storage Analytics metrics, also referred to as *classic metrics* will be retired. For more information, see the [official announcement](#). If you use classic metrics, make sure to transition to metrics in Azure Monitor prior to that date. This article helps you make the transition.

## Steps to complete the transition

To transition to metrics in Azure Monitor, we recommend the following approach.

1. Learn about some of the [key differences](#) between classic metrics and metrics in Azure Monitor.
2. Compile a list of classic metrics that you currently use.
3. Identify [which metrics in Azure Monitor](#) provide the same data as the metrics you currently use.
4. Create [charts](#) or [dashboards](#) to view metric data.

### NOTE

Metrics in Azure Monitor are enabled by default, so there is nothing you need to do to begin capturing metrics. You must however, create charts or dashboards to view those metrics.

5. If you've created alert rules that are based on classic storage metrics, then [create alert rules](#) that are based on metrics in Azure Monitor.
6. After you're able to see all of your metrics in Azure Monitor, you can turn off classic logging.

## Classic metrics vs. metrics in Azure Monitor

This section describes a few key differences between these two metrics platforms.

The main difference is in how metrics are managed. Classic metrics are managed by Azure Storage whereas metrics in Azure Monitor are managed by Azure Monitor. With classic metrics, Azure Storage collects metric values, aggregates them, and then stores them in tables that are located in the storage account. With metrics in Azure Monitor, Azure Storage sends metric data to the Azure Monitor back end. Azure Monitor provides a unified monitoring experience that includes data from the Azure portal as well as data that is ingested.

Classic metrics are sent and stored in an Azure storage account. Azure Monitor metrics can be sent to multiple locations. A storage account can be one of those locations, but it is not required.

As far as metrics support, classic metrics provide **capacity** metrics only for Azure Blob storage. Metrics in Azure Monitor provide capacity metrics for Blob, Table, File, Queue, and premium storage. Classic metrics provide **transaction** metrics on Blob, Table, Azure File, and Queue storage. Metrics in Azure Monitor add premium storage to that list.

If the activity in your account does not trigger a metric, classic metrics will show a value of zero (0) for that metric. Metrics in Azure Monitor will omit the data entirely, which leads to cleaner reports. For example, with classic metrics, if no server timeout errors are reported, then the `serverTimeoutError` value in the metrics table is set to 0. Azure Monitor doesn't return any data when you query the value of metric `Transactions` with dimension `ResponseType` equal to `ServerTimeoutError`.

To learn more about metrics in Azure Monitor, see [Metrics in Azure Monitor](#).

## Map classic metrics to metrics in Azure Monitor

Use these tables to identify which metrics in Azure Monitor provide the same data as the metrics you currently use.

### Capacity metrics

CLASSIC METRIC	METRIC IN AZURE MONITOR
Capacity	BlobCapacity with the dimension BlobType equal to BlockBlob or PageBlob
ObjectCount	BlobCount with the dimension BlobType equal to BlockBlob or PageBlob
ContainerCount	ContainerCount

#### NOTE

There are also several new capacity metrics that weren't available as classic metrics. To view the complete list, see [Metrics](#).

### Transaction metrics

CLASSIC METRIC	METRIC IN AZURE MONITOR
AnonymousAuthorizationError	Transactions with the dimension ResponseType equal to AuthorizationError and dimension Authentication equal to Anonymous
AnonymousClientOtherError	Transactions with the dimension ResponseType equal to ClientOtherError and dimension Authentication equal to Anonymous
AnonymousClientTimeoutError	Transactions with the dimension ResponseType equal to ClientTimeoutError and dimension Authentication equal to Anonymous
AnonymousNetworkError	Transactions with the dimension ResponseType equal to NetworkError and dimension Authentication equal to Anonymous
AnonymousServerOtherError	Transactions with the dimension ResponseType equal to ServerOtherError and dimension Authentication equal to Anonymous
AnonymousServerTimeoutError	Transactions with the dimension ResponseType equal to ServerTimeoutError and dimension Authentication equal to Anonymous

CLASSIC METRIC	METRIC IN AZURE MONITOR
AnonymousSuccess	Transactions with the dimension <code>ResponseType</code> equal to <code>Success</code> and dimension <code>Authentication</code> equal to <code>Anonymous</code>
AnonymousThrottlingError	Transactions with the dimension <code>ResponseType</code> equal to <code>ClientThrottlingError</code> or <code>ServerBusyError</code> and dimension <code>Authentication</code> equal to <code>Anonymous</code>
AuthorizationError	Transactions with the dimension <code>ResponseType</code> equal to <code>AuthorizationError</code>
Availability	Availability
AverageE2ELatency	SuccessE2ELatency
AverageServerLatency	SuccessServerLatency
ClientOtherError	Transactions with the dimension <code>ResponseType</code> equal to <code>ClientOtherError</code>
ClientTimeoutError	Transactions with the dimension <code>ResponseType</code> equal to <code>ClientTimeoutError</code>
NetworkError	Transactions with the dimension <code>ResponseType</code> equal to <code>NetworkError</code>
PercentAuthorizationError	Transactions with the dimension <code>ResponseType</code> equal to <code>AuthorizationError</code>
PercentClientOtherError	Transactions with the dimension <code>ResponseType</code> equal to <code>ClientOtherError</code>
PercentNetworkError	Transactions with the dimension <code>ResponseType</code> equal to <code>NetworkError</code>
PercentServerOtherError	Transactions with the dimension <code>ResponseType</code> equal to <code>ServerOtherError</code>
PercentSuccess	Transactions with the dimension <code>ResponseType</code> equal to <code>Success</code>
PercentThrottlingError	Transactions with the dimension <code>ResponseType</code> equal to <code>ClientThrottlingError</code> or <code>ServerBusyError</code>
PercentTimeoutError	Transactions with the dimension <code>ResponseType</code> equal to <code>ServerError</code> or <code>ResponseType</code> equal to <code>ClientTimeoutError</code>
SASAuthorizationError	Transactions with the dimension <code>ResponseType</code> equal to <code>AuthorizationError</code> and dimension <code>Authentication</code> equal to <code>SAS</code>

CLASSIC METRIC	METRIC IN AZURE MONITOR
SASClientOtherError	Transactions with the dimension <code>ResponseType</code> equal to <code>ClientOtherError</code> and dimension <code>Authentication</code> equal to <code>SAS</code>
SASClientTimeoutError	Transactions with the dimension <code>ResponseType</code> equal to <code>ClientTimeoutError</code> and dimension <code>Authentication</code> equal to <code>SAS</code>
SASNetworkError	Transactions with the dimension <code>ResponseType</code> equal to <code>NetworkError</code> and dimension <code>Authentication</code> equal to <code>SAS</code>
SASServerOtherError	Transactions with the dimension <code>ResponseType</code> equal to <code>ServerOtherError</code> and dimension <code>Authentication</code> equal to <code>SAS</code>
SASServerTimeoutError	Transactions with the dimension <code>ResponseType</code> equal to <code>ServerTimeoutError</code> and dimension <code>Authentication</code> equal to <code>SAS</code>
SASSuccess	Transactions with the dimension <code>ResponseType</code> equal to <code>Success</code> and dimension <code>Authentication</code> equal to <code>SAS</code>
SASThrottlingError	Transactions with the dimension <code>ResponseType</code> equal to <code>ClientThrottlingError</code> or <code>ServerBusyError</code> and dimension <code>Authentication</code> equal to <code>SAS</code>
ServerOtherError	Transactions with the dimension <code>ResponseType</code> equal to <code>ServerOtherError</code>
ServerTimeoutError	Transactions with the dimension <code>ResponseType</code> equal to <code>ServerTimeoutError</code>
Success	Transactions with the dimension <code>ResponseType</code> equal to <code>Success</code>
ThrottlingError	Transactions with the dimension <code>ResponseType</code> equal to <code>ClientThrottlingError</code> or <code>ServerBusyError</code>
TotalBillableRequests	Transactions
TotalEgress	Egress
TotalIngress	Ingress
TotalRequests	Transactions

## Next steps

- [Azure Monitor](#)

# Storage Analytics

8/22/2022 • 2 minutes to read • [Edit Online](#)

Azure Storage Analytics performs logging and provides metrics data for a storage account. You can use this data to trace requests, analyze usage trends, and diagnose issues with your storage account.

To use Storage Analytics, you must enable it individually for each service you want to monitor. You can enable it from the [Azure portal](#). For details, see [Monitor a storage account in the Azure portal](#). You can also enable Storage Analytics programmatically via the REST API or the client library. Use the [Set Blob Service Properties](#), [Set Queue Service Properties](#), [Set Table Service Properties](#), and [Set File Service Properties](#) operations to enable Storage Analytics for each service.

The aggregated data is stored in a well-known blob (for logging) and in well-known tables (for metrics), which may be accessed using the Blob service and Table service APIs.

Storage Analytics has a 20 TB limit on the amount of stored data that is independent of the total limit for your storage account. For more information about storage account limits, see [Scalability and performance targets for standard storage accounts](#).

For an in-depth guide on using Storage Analytics and other tools to identify, diagnose, and troubleshoot Azure Storage-related issues, see [Monitor, diagnose, and troubleshoot Microsoft Azure Storage](#).

## Billing for Storage Analytics

All metrics data is written by the services of a storage account. As a result, each write operation performed by Storage Analytics is billable. Additionally, the amount of storage used by metrics data is also billable.

The following actions performed by Storage Analytics are billable:

- Requests to create blobs for logging.
- Requests to create table entities for metrics.

If you have configured a data retention policy, you can reduce the spending by deleting old logging and metrics data. For more information about retention policies, see [Setting a Storage Analytics Data Retention Policy](#).

### Understanding billable requests

Every request made to an account's storage service is either billable or non-billable. Storage Analytics logs each individual request made to a service, including a status message that indicates how the request was handled. Similarly, Storage Analytics stores metrics for both a service and the API operations of that service, including the percentages and count of certain status messages. Together, these features can help you analyze your billable requests, make improvements on your application, and diagnose issues with requests to your services. For more information about billing, see [Understanding Azure Storage Billing - Bandwidth, Transactions, and Capacity](#).

When looking at Storage Analytics data, you can use the tables in the [Storage Analytics Logged Operations and Status Messages](#) topic to determine what requests are billable. Then you can compare your logs and metrics data to the status messages to see if you were charged for a particular request. You can also use the tables in the previous topic to investigate availability for a storage service or individual API operation.

## Next steps

- [Monitor a storage account in the Azure portal](#)
- [Storage Analytics Metrics](#)

- Storage Analytics Logging

# Azure Storage Analytics metrics (classic)

8/22/2022 • 5 minutes to read • [Edit Online](#)

On August 31, 2023 Storage Analytics metrics, also referred to as *classic metrics* will be retired. For more information, see the [official announcement](#). If you use classic metrics, make sure to transition to metrics in Azure Monitor prior to that date. This article helps you make the transition.

Azure Storage uses the Storage Analytics solution to store metrics that include aggregated transaction statistics and capacity data about requests to a storage service. Transactions are reported at the API operation level and at the storage service level. Capacity is reported at the storage service level. Metrics data can be used to:

- Analyze storage service usage.
- Diagnose issues with requests made against the storage service.
- Improve the performance of applications that use a service.

Storage Analytics metrics are enabled by default for new storage accounts. You can configure metrics in the [Azure portal](#), by using PowerShell, or by using the Azure CLI. For step-by-step guidance, see [Enable and manage Azure Storage Analytic metrics \(classic\)](#). You can also enable Storage Analytics programmatically via the REST API or the client library. Use the Set Service Properties operations to enable Storage Analytics for each service.

## NOTE

Storage Analytics metrics are available for Azure Blob storage, Azure Queue storage, Azure Table storage, and Azure Files. Storage Analytics metrics are now classic metrics. We recommend that you use [storage metrics in Azure Monitor](#) instead of Storage Analytics metrics.

## Transaction metrics

A robust set of data is recorded at hourly or minute intervals for each storage service and requested API operation, which includes ingress and egress, availability, errors, and categorized request percentages. For a complete list of the transaction details, see [Storage Analytics metrics table schema](#).

Transaction data is recorded at the service level and the API operation level. At the service level, statistics that summarize all requested API operations are written to a table entity every hour, even if no requests were made to the service. At the API operation level, statistics are only written to an entity if the operation was requested within that hour.

For example, if you perform a **GetBlob** operation on your blob service, Storage Analytics Metrics logs the request and includes it in the aggregated data for the blob service and the **GetBlob** operation. If no **GetBlob** operation is requested during the hour, an entity isn't written to **\$MetricsTransactionsBlob** for that operation.

Transaction metrics are recorded for user requests and requests made by Storage Analytics itself. For example, requests by Storage Analytics to write logs and table entities are recorded.

## Capacity metrics

## NOTE

Currently, capacity metrics are available only for the blob service.

Capacity data is recorded daily for a storage account's blob service, and two table entities are written. One entity provides statistics for user data, and the other provides statistics about the `$logs` blob container used by Storage Analytics. The `$MetricsCapacityBlob` table includes the following statistics:

- **Capacity**: The amount of storage used by the storage account's blob service, in bytes.
- **ContainerCount**: The number of blob containers in the storage account's blob service.
- **ObjectCount**: The number of committed and uncommitted block or page blobs in the storage account's blob service.

For more information about capacity metrics, see [Storage Analytics metrics table schema](#).

## How metrics are stored

All metrics data for each of the storage services is stored in three tables reserved for that service. One table is for transaction information, one table is for minute transaction information, and another table is for capacity information. Transaction and minute transaction information consists of request and response data. Capacity information consists of storage usage data. Hour metrics, minute metrics, and capacity for a storage account's blob service is accessed in tables that are named as described in the following table.

METRICS LEVEL	TABLE NAMES	SUPPORTED FOR VERSIONS
Hourly metrics, primary location	- <code>\$MetricsTransactionsBlob</code> - <code>\$MetricsTransactionsTable</code> - <code>\$MetricsTransactionsQueue</code>	Versions prior to August 15, 2013, only. While these names are still supported, we recommend that you switch to using the tables that follow.
Hourly metrics, primary location	- <code>\$MetricsHourPrimaryTransactionsBlob</code> - <code>\$MetricsHourPrimaryTransactionsTable</code> - <code>\$MetricsHourPrimaryTransactionsQueue</code> - <code>\$MetricsHourPrimaryTransactionsFile</code>	All versions. Support for file service metrics is available only in version April 5, 2015, and later.
Minute metrics, primary location	- <code>\$MetricsMinutePrimaryTransactionsBlob</code> - <code>\$MetricsMinutePrimaryTransactionsTable</code> - <code>\$MetricsMinutePrimaryTransactionsQueue</code> - <code>\$MetricsMinutePrimaryTransactionsFile</code>	All versions. Support for file service metrics is available only in version April 5, 2015, and later.
Hourly metrics, secondary location	- <code>\$MetricsHourSecondaryTransactionsBlob</code> - <code>\$MetricsHourSecondaryTransactionsTable</code> - <code>\$MetricsHourSecondaryTransactionsQueue</code>	All versions. Read-access geo-redundant replication must be enabled.

METRICS LEVEL	TABLE NAMES	SUPPORTED FOR VERSIONS
Minute metrics, secondary location	- \$MetricsMinuteSecondaryTransactions Blob - \$MetricsMinuteSecondaryTransactions Table - \$MetricsMinuteSecondaryTransactions Queue	All versions. Read-access geo-redundant replication must be enabled.
Capacity (blob service only)	\$MetricsCapacityBlob	All versions.

These tables are automatically created when Storage Analytics is enabled for a storage service endpoint. They're accessed via the namespace of the storage account, for example,

`https://<accountname>.table.core.windows.net/Tables("$MetricsTransactionsBlob")`. The metrics tables don't appear in a listing operation and must be accessed directly via the table name.

## Metrics alerts

Consider setting up alerts in the [Azure portal](#) so you'll be automatically notified of important changes in the behavior of your storage services. For step-by-step guidance, see [Create metrics alerts](#).

If you use a Storage Explorer tool to download this metrics data in a delimited format, you can use Microsoft Excel to analyze the data. For a list of available Storage Explorer tools, see [Azure Storage client tools](#).

### IMPORTANT

There might be a delay between a storage event and when the corresponding hourly or minute metrics data is recorded. In the case of minute metrics, several minutes of data might be written at once. This issue can lead to transactions from earlier minutes being aggregated into the transaction for the current minute. When this issue happens, the alert service might not have all available metrics data for the configured alert interval, which might lead to alerts firing unexpectedly.

## Billing on storage metrics

Write requests to create table entities for metrics are charged at the standard rates applicable to all Azure Storage operations.

Read requests of metrics data by a client are also billable at standard rates.

The capacity used by the metrics tables is also billable. Use the following information to estimate the amount of capacity used for storing metrics data:

- If each hour a service utilizes every API in every service, approximately 148 KB of data is stored every hour in the metrics transaction tables if you enabled a service-level and API-level summary.
- If within each hour a service utilizes every API in the service, approximately 12 KB of data is stored every hour in the metrics transaction tables if you enabled only a service-level summary.
- The capacity table for blobs has two rows added each day provided you opted in for logs. This scenario implies that every day the size of this table increases by up to approximately 300 bytes.

## Next steps

- [Storage Analytics metrics table schema](#)
- [Storage Analytics logged operations and status messages](#)

- Storage Analytics logging

# Azure Storage analytics logging

8/22/2022 • 6 minutes to read • [Edit Online](#)

Storage Analytics logs detailed information about successful and failed requests to a storage service. This information can be used to monitor individual requests and to diagnose issues with a storage service. Requests are logged on a best-effort basis. This means that most requests will result in a log record, but the completeness and timeliness of Storage Analytics logs are not guaranteed.

## NOTE

We recommend that you use Azure Storage logs in Azure Monitor instead of Storage Analytics logs. To learn more, see any of the following articles:

- [Monitoring Azure Blob Storage](#)
- [Monitoring Azure Files](#)
- [Monitoring Azure Queue Storage](#)
- [Monitoring Azure Table storage](#)

Storage Analytics logging is not enabled by default for your storage account. You can enable it in the [Azure portal](#) or by using PowerShell, or Azure CLI. For step-by-step guidance, see [Enable and manage Azure Storage Analytics logs \(classic\)](#).

You can also enable Storage Analytics logs programmatically via the REST API or the client library. Use the [Get Blob Service Properties](#), [Get Queue Service Properties](#), and [Get Table Service Properties](#) operations to enable Storage Analytics for each service. To see an example that enables Storage Analytics logs by using .NET, see [Enable logs](#)

Log entries are created only if there are requests made against the service endpoint. For example, if a storage account has activity in its Blob endpoint but not in its Table or Queue endpoints, only logs pertaining to the Blob service will be created.

## NOTE

Storage Analytics logging is currently available only for the Blob, Queue, and Table services. Storage Analytics logging is also available for premium-performance [BlockBlobStorage](#) accounts. However, it isn't available for general-purpose v2 accounts with premium performance.

## Requests logged in logging

### Logging authenticated requests

The following types of authenticated requests are logged:

- Successful requests
- Failed requests, including timeout, throttling, network, authorization, and other errors
- Requests using a Shared Access Signature (SAS) or OAuth, including failed and successful requests
- Requests to analytics data

Requests made by Storage Analytics itself, such as log creation or deletion, are not logged. A full list of the logged data is documented in the [Storage Analytics Logged Operations and Status Messages](#) and [Storage Analytics Log Format](#) topics.

### Logging anonymous requests

The following types of anonymous requests are logged:

- Successful requests
- Server errors
- Timeout errors for both client and server
- Failed GET requests with error code 304 (Not Modified)

All other failed anonymous requests are not logged. A full list of the logged data is documented in the [Storage Analytics Logged Operations and Status Messages](#) and [Storage Analytics Log Format](#) topics.

## NOTE

Storage Analytics logs all internal calls to the data plane. Calls from the Azure Storage Resource Provider are also logged. To identify these requests, look for the query string `<sk=system-1>` in the request URL.

## How logs are stored

All logs are stored in block blobs in a container named `$logs`, which is automatically created when Storage Analytics is enabled for a storage account. The `$logs` container is located in the blob namespace of the storage account, for example: [http://<accountname>.blob.core.windows.net/\\$logs](http://<accountname>.blob.core.windows.net/$logs). This container cannot be deleted once Storage Analytics has been enabled, though its contents can be deleted. If you use your storage-browsing tool to navigate to the container directly, you will see all the blobs that contain your logging data.

**NOTE**

The `$logs` container is not displayed when a container listing operation is performed, such as the List Containers operation. It must be accessed directly. For example, you can use the List Blobs operation to access the blobs in the `$logs` container.

As requests are logged, Storage Analytics will upload intermediate results as blocks. Periodically, Storage Analytics will commit these blocks and make them available as a blob. It can take up to an hour for log data to appear in the blobs in the `$logs` container because the frequency at which the storage service flushes the log writers. Duplicate records may exist for logs created in the same hour. You can determine if a record is a duplicate by checking the `RequestId` and `Operation` number.

If you have a high volume of log data with multiple files for each hour, then you can use the blob metadata to determine what data the log contains by examining the blob metadata fields. This is also useful because there can sometimes be a delay while data is written to the log files: the blob metadata gives a more accurate indication of the blob content than the blob name.

Most storage browsing tools enable you to view the metadata of blobs; you can also read this information using PowerShell or programmatically. The following PowerShell snippet is an example of filtering the list of log blobs by name to specify a time, and by metadata to identify just those logs that contain `write` operations.

```
Get-AzStorageBlob -Container '$logs' |
Where-Object {
 $_.Name -match 'blob/2014/05/21/05' -and
 $_.ICloudBlob.Metadata.LogType -match 'write'
} |
ForEach-Object {
 "{0} {1} {2} {3}" -f $_.Name,
 $_.ICloudBlob.Metadata.StartTime,
 $_.ICloudBlob.Metadata.EndTime,
 $_.ICloudBlob.Metadata.LogType
}
```

For information about listing blobs programmatically, see [Enumerating Blob Resources](#) and [Setting and Retrieving Properties and Metadata for Blob Resources](#).

#### Log naming conventions

Each log will be written in the following format:

```
<service-name>/YYYY/MM/DD/hhmm/<counter>.log
```

The following table describes each attribute in the log name:

ATTRIBUTE	DESCRIPTION
<code>&lt;service-name&gt;</code>	The name of the storage service. For example: <code>blob</code> , <code>table</code> , or <code>queue</code>
<code>YYYY</code>	The four digit year for the log. For example: <code>2011</code>
<code>MM</code>	The two digit month for the log. For example: <code>07</code>
<code>DD</code>	The two digit day for the log. For example: <code>31</code>
<code>hh</code>	The two digit hour that indicates the starting hour for the logs, in 24 hour UTC format. For example: <code>18</code>
<code>mm</code>	The two digit number that indicates the starting minute for the logs. <b>Note:</b> This value is unsupported in the current version of Storage Analytics, and its value will always be <code>00</code> .
<code>&lt;counter&gt;</code>	A zero-based counter with six digits that indicates the number of log blobs generated for the storage service in an hour time period. This counter starts at <code>000000</code> . For example: <code>000001</code>

The following is a complete sample log name that combines the above examples:

```
blob/2011/07/31/1800/000001.log
```

The following is a sample URI that can be used to access the above log:

```
https://<accountname>.blob.core.windows.net/$logs/blob/2011/07/31/1800/000001.log
```

When a storage request is logged, the resulting log name correlates to the hour when the requested operation completed. For example, if a GetBlob request was completed at 6:30PM on 7/31/2011, the log would be written with the following prefix: `blob/2011/07/31/1800/`

### Log metadata

All log blobs are stored with metadata that can be used to identify what logging data the blob contains. The following table describes each metadata attribute:

ATTRIBUTE	DESCRIPTION
<code>LogType</code>	Describes whether the log contains information pertaining to read, write, or delete operations. This value can include one type or a combination of all three, separated by commas.  Example 1: <code>write</code>  Example 2: <code>read,write</code>  Example 3: <code>read,write,delete</code>
<code>StartTime</code>	The earliest time of an entry in the log, in the form of <code>YYYY-MM-DDThh:mm:ssZ</code> . For example:  <code>2011-07-31T18:21:46Z</code>
<code>EndTime</code>	The latest time of an entry in the log, in the form of <code>YYYY-MM-DDThh:mm:ssZ</code> . For example:  <code>2011-07-31T18:22:09Z</code>
<code>LogVersion</code>	The version of the log format.

The following list displays complete sample metadata using the above examples:

- `LogType=write`
- `StartTime=2011-07-31T18:21:46Z`
- `EndTime=2011-07-31T18:22:09Z`
- `LogVersion=1.0`

### Log entries

The following sections show an example log entry for each supported Azure Storage service.

#### Example log entry for Blob Storage

```
2.0;2022-01-03T0:34:54.4617505Z;PutBlob;SASSuccess;201;7;7;sas;;logsamples/blob;https://logsamples.blob.core.windows.net/container1/1.txt?se=2022-02-02T20:34:54Z&sig=XXXXXX&sp=rw&sr=c&sv=2020-04-08&timeout=901;" /logsamples/container1/1.txt";xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx;0;71.197.193.44:53371;2019-12-12;654;13;37;0;13;"xxxxxxxxxxxxxxxxxxxxx=";"xxxxxxxxxxxxxxxxxxxxx=";""0x8D9CEF88004E296"" ;Monday, 03-Jan-22 20:34:54 GMT;"Microsoft Azure Storage Explorer, 1.20.1, win32, azcopy-node, 2.0.0, win32, AzCopy/10.11.0 Azure-Storage/0.13 (go1.15; Windows_NT) ";"xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxx";;;;;;
```

#### Example log entry for Blob Storage (Data Lake Storage Gen2 enabled)

```
2.0;2022-01-04T22:50:56.0000775Z;RenamePathFile;Success;201;49;49;authenticated;logsamples;logsamples/blob;"https://logsamples.dfs.core.windows.net/my-container/myfileorig.png?mode=legacy";"/logsamples/my-container/myfilerenamed.png";xxxxxxxx-xxxx-xxxx-xxxxxxxxxx;0;73.157.16.8;2020-04-08;591;0;224;0;0;;;;Friday, 11-Jun-21 17:58:15 GMT;"Microsoft Azure Storage Explorer, 1.19.1, win32 azsdk-js-storagedatalake/12.3.1 (NODE-VERSION v12.16.3; Windows_NT 10.0.22000) ";"xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxx";;;;;;
```

#### Example log entry for Queue Storage

```
2.0;2022-01-03T20:35:04.6097590Z;PeekMessages;Success;200;5;authenticated;logsamples;logsamples;queue;https://logsamples.queue.core.windows.net/queue1/messages;numofmessages=32&peekonly=true&timeout=30;" /logsamples/queue1";xxxxxxxx-xxxx-xxxx-xxxxxxxxxx;0;71.197.193.44:53385;2020-04-08;536;0;232;62;0;;;;;"Microsoft Azure Storage Explorer, 1.20.1, win32 azsdk-js-storagequeue/12.3.1 (NODE-VERSION v12.16.3; Windows_NT 10.0.22000) ";"xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxx";;;;;;
```

#### Example log entry for Table Storage

```
1.0;2022-01-03T20:35:13.0719766Z;CreateTable;Success;204;30;30;authenticated;logsamples;logsamples;table;https://logsamples.table.core.windows.net/Tables;" /logxxxx-xxxx-xxxx-xxxxxxxxxx;0;71.197.193.44:53389;2018-03-28;601;22;339;0;22;;;;;"Microsoft Azure Storage Explorer, 1.20.1, win32, Azure-Storage/2.12.16.3; Windows_NT 10.0.22000) ";"xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxx"
```

## Next steps

- [Enable and manage Azure Storage Analytics logs \(classic\)](#)
- [Storage Analytics Log Format](#)
- [Storage Analytics Logged Operations and Status Messages](#)
- [Storage Analytics Metrics \(classic\)](#)

# Network File System (NFS) 3.0 protocol support for Azure Blob Storage

8/22/2022 • 4 minutes to read • [Edit Online](#)

Blob storage now supports the Network File System (NFS) 3.0 protocol. This support provides Linux file system compatibility at object storage scale and prices and enables Linux clients to mount a container in Blob storage from an Azure Virtual Machine (VM) or a computer on-premises.

It's always been a challenge to run large-scale legacy workloads, such as High Performance Computing (HPC) in the cloud. One reason is that applications often use traditional file protocols such as NFS or Server Message Block (SMB) to access data. Also, native cloud storage services focused on object storage that have a flat namespace and extensive metadata instead of file systems that provide a hierarchical namespace and efficient metadata operations.

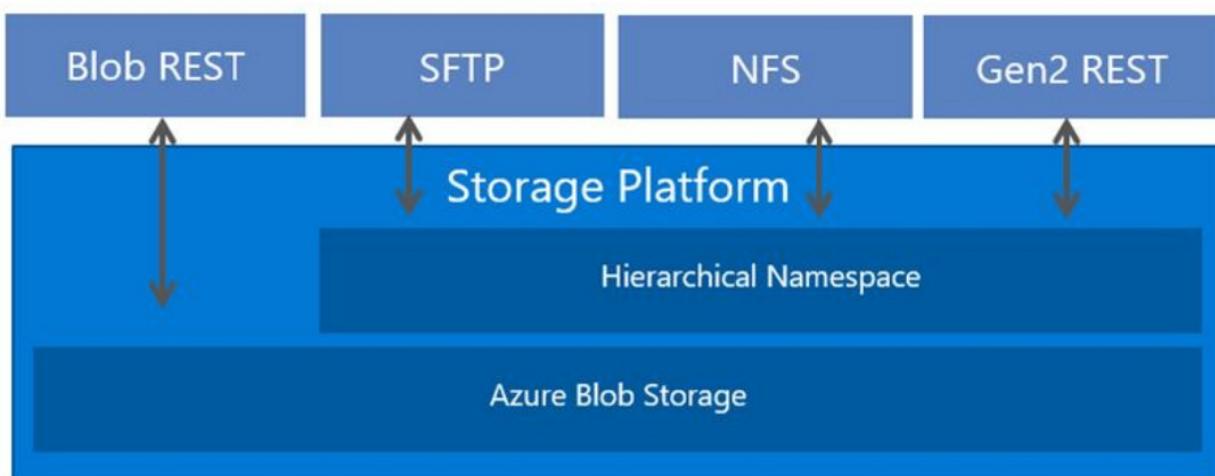
Blob Storage now supports a hierarchical namespace, and when combined with NFS 3.0 protocol support, Azure makes it much easier to run legacy applications on top of large-scale cloud object storage.

## Applications and workloads suited for this feature

The NFS 3.0 protocol feature is best suited for processing high throughput, high scale, read heavy workloads such as media processing, risk simulations, and genomics sequencing. You should consider using this feature for any other type of workload that uses multiple readers and many threads, which require high bandwidth.

## NFS 3.0 and the hierarchical namespace

NFS 3.0 protocol support requires blobs to be organized into a hierarchical namespace. You can enable a hierarchical namespace when you create a storage account. The ability to use a hierarchical namespace was introduced by Azure Data Lake Storage Gen2. It organizes objects (files) into a hierarchy of directories and subdirectories in the same way that the file system on your computer is organized. The hierarchical namespace scales linearly and doesn't degrade data capacity or performance. Different protocols extend from the hierarchical namespace. The NFS 3.0 protocol is one of the available protocols.



## Data stored as block blobs

If you enable NFS 3.0 protocol support, all of the data in your storage account will be stored as block blobs. Block blobs are optimized to efficiently process large amounts of read-heavy data. Block blobs are composed of

blocks. Each block is identified by a block ID. A block blob can include up to 50,000 blocks. Each block in a block blob can be a different size, up to the maximum size permitted for the service version that your account uses.

When your application makes a request by using the NFS 3.0 protocol, that request is translated into combination of block blob operations. For example, NFS 3.0 read Remote Procedure Call (RPC) requests are translated into [Get Blob](#) operation. NFS 3.0 write RPC requests are translated into a combination of [Get Block List](#), [Put Block](#), and [Put Block List](#).

## General workflow: Mounting a storage account container

Your Linux clients can mount a container in Blob storage from an Azure Virtual Machine (VM) or a computer on-premises. To mount a storage account container, you'll have to do these things.

1. Create an Azure Virtual Network (VNet).
2. Configure network security.
3. Create and configure storage account that accepts traffic only from the VNet.
4. Create a container in the storage account.
5. Mount the container.

For step-by-step guidance, see [Mount Blob storage by using the Network File System \(NFS\) 3.0 protocol](#).

## Network security

Traffic must originate from a VNet. A VNet enables clients to securely connect to your storage account. The only way to secure the data in your account is by using a VNet and other network security settings. Any other tool used to secure data including account key authorization, Azure Active Directory (AD) security, and access control lists (ACLs) are not yet supported in accounts that have the NFS 3.0 protocol support enabled on them.

To learn more, see [Network security recommendations for Blob storage](#).

### Supported network connections

A client can connect over a public or a [private endpoint](#), and can connect from any of the following network locations:

- The VNet that you configure for your storage account.

In this article, we'll refer to that VNet as the *primary VNet*. To learn more, see [Grant access from a virtual network](#).

- A peered VNet that is in the same region as the primary VNet.

You'll have to configure your storage account to allow access to this peered VNet. To learn more, see [Grant access from a virtual network](#).

- An on-premises network that is connected to your primary VNet by using [VPN Gateway](#) or an [ExpressRoute gateway](#).

To learn more, see [Configuring access from on-premises networks](#).

- An on-premises network that is connected to a peered network.

This can be done by using [VPN Gateway](#) or an [ExpressRoute gateway](#) along with [Gateway transit](#).

**IMPORTANT**

If you're connecting from an on-premises network, make sure that your client allows outgoing communication through ports 111 and 2048. The NFS 3.0 protocol uses these ports.

## Known issues and limitations

See the [Known issues](#) article for a complete list of issues and limitations with the current release of NFS 3.0 support.

## Pricing

See the [Azure Blob Storage pricing](#) page for data storage and transaction costs.

## See also

- [Mount Blob storage by using the Network File System \(NFS\) 3.0 protocol](#)
- [Network File System \(NFS\) 3.0 performance considerations in Azure Blob Storage](#)
- [Compare access to Azure Files, Blob Storage, and Azure NetApp Files with NFS](#)

# Network File System (NFS) 3.0 performance considerations in Azure Blob storage

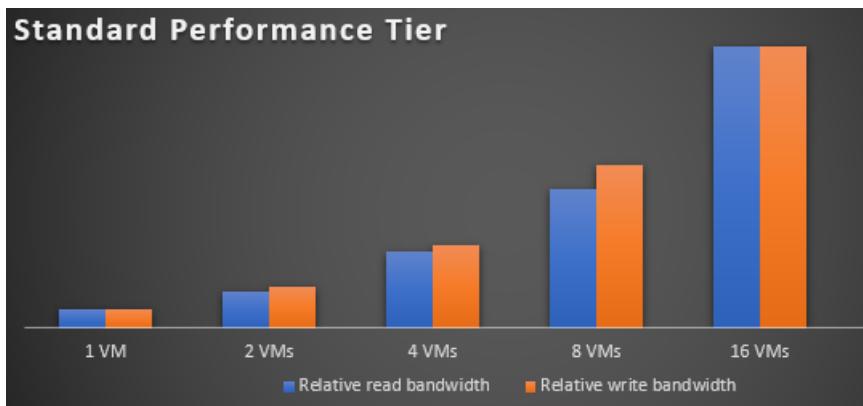
8/22/2022 • 2 minutes to read • [Edit Online](#)

Blob storage now supports the Network File System (NFS) 3.0 protocol. This article contains recommendations that help you to optimize the performance of your storage requests. To learn more about NFS 3.0 support for Azure Blob Storage, see [Network File System \(NFS\) 3.0 protocol support for Azure Blob storage](#).

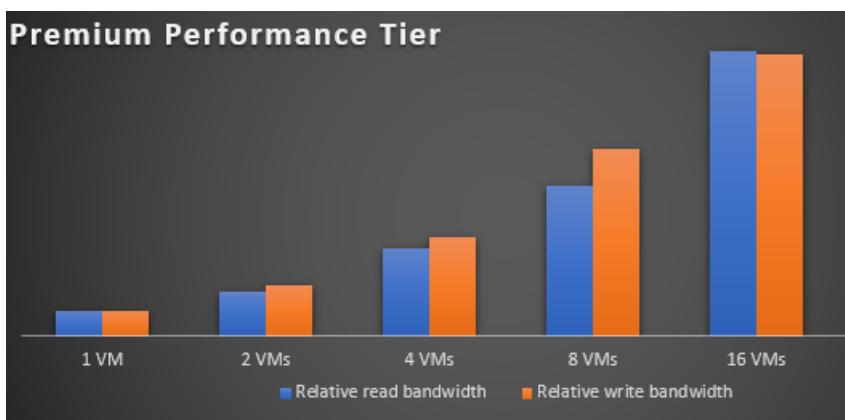
## Add clients to increase throughput

Azure Blob Storage scales linearly until it reaches the maximum storage account egress and ingress limit. Therefore, your applications can achieve higher throughput by using more clients. To view storage account egress and ingress limits, see [Scalability and performance targets for standard storage accounts](#).

The following chart shows how bandwidth increases as you add more clients. In this chart, a client is a Virtual Machine (VM) and with a standard general-purpose v2 storage account.



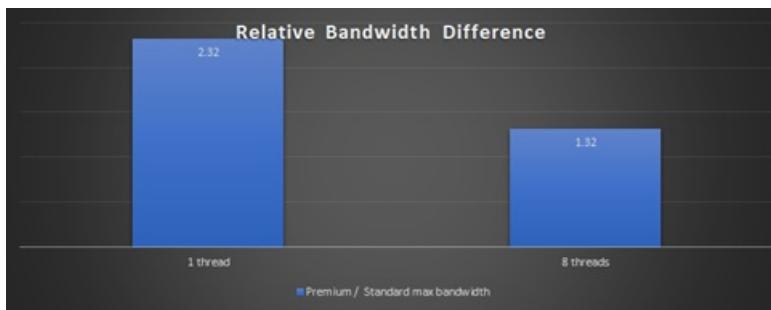
The following chart shows this same effect when applied to a premium block blob storage account.



## Use premium block blob storage accounts for small scale applications

Not all applications can scale up by adding more clients. For those applications, [Azure premium block blob storage account](#) offers consistent low-latency and high transaction rates. The premium block blob storage account can reach maximum bandwidth with fewer threads and clients. For example, with a single client, a premium block blob storage account can achieve 2.3x bandwidth compared to the same setup used with a standard performance general purpose v2 storage account.

Each bar in the following chart shows the difference in achieved bandwidth between premium and standard performance storage accounts. As the number of clients increases, that difference decreases.



## Improve read ahead size to increase large file read throughput

The `read_ahead_kb` kernel parameter represents the amount of additional data that should be read after fulfilling a given read request. You can increase this parameter to 16 MiB to improve large file read throughput.

```
export AZMNT=/your/container/mountpoint
echo 16384 > /sys/class/bdi/0:$((stat -c "%d" $AZMNT)/read_ahead_kb)
```

## Avoid frequent overwrites on data

It takes longer time to complete an overwrite operation than a new write operation. That's because an NFS overwrite operation, especially a partial in-place file edit, is a combination of several underlying blob operations: a read, a modify, and a write operation. Therefore, an application that requires frequent in place edits is not suited for NFS enabled blob storage accounts.

## Deploy Azure HPC Cache for latency sensitive applications

Some applications may require low latency in addition to high throughput. You can deploy [Azure HPC Cache](#) to improve latency significantly. Learn more about [Latency in Blob storage](#).

## Other best practice recommendations

- Use VMs with sufficient network bandwidth.
- Use multiple mount points when your workloads allow it.
- Use as many threads as possible.
- Use large block sizes.
- Make storage requests from a client that is located in the same region as the storage account. This can improve network latency.

## Next steps

- To learn more about NFS 3.0 support for Azure Blob Storage, see [Network File System \(NFS\) 3.0 protocol support for Azure Blob storage](#).
- To get started, see [Mount Blob storage by using the Network File System \(NFS\) 3.0 protocol](#).

# Known issues with Network File System (NFS) 3.0 protocol support for Azure Blob Storage

8/22/2022 • 2 minutes to read • [Edit Online](#)

This article describes limitations and known issues of Network File System (NFS) 3.0 protocol support for Azure Blob Storage.

## IMPORTANT

Because you must enable the hierarchical namespace feature of your account to use NFS 3.0, all of the known issues that are described in the [Known issues with Azure Data Lake Storage Gen2](#) article also apply to your account.

## NFS 3.0 support

- NFS 3.0 support can't be enabled on existing storage accounts.
- NFS 3.0 support can't be disabled in a storage account after you've enabled it.
- GRS, GZRS, and RA-GRS redundancy options aren't supported when you create an NFS 3.0 storage account.

## NFS 3.0 features

The following NFS 3.0 features aren't yet supported.

- NFS 3.0 over UDP. Only NFS 3.0 over TCP is supported.
- Locking files with Network Lock Manager (NLM). Mount commands must include the `-o nolock` parameter.
- Mounting subdirectories. You can only mount the root directory (Container).
- Listing mounts (For example: by using the command `showmount -a`)
- Listing exports (For example: by using the command `showmount -e`)
- Hard link
- Exporting a container as read-only

## NFS 3.0 clients

Windows client for NFS is not yet supported

## Blob Storage features

When you enable NFS 3.0 protocol support, some Blob Storage features will be fully supported, but some features might be supported only at the preview level or not yet supported at all.

To see how each Blob Storage feature is supported in accounts that have NFS 3.0 support enabled, see [Blob Storage feature support for Azure Storage accounts](#).

**NOTE**

Static websites is an example of a partially supported feature because the configuration page for static websites does not yet appear in the Azure portal for accounts that have NFS 3.0 support enabled. You can enable static websites only by using PowerShell or Azure CLI.

## See also

- [Network File System \(NFS\) 3.0 protocol support for Azure Blob Storage](#)
- [Mount Blob storage by using the Network File System \(NFS\) 3.0 protocol](#)

# SSH File Transfer Protocol (SFTP) support for Azure Blob Storage (preview)

8/22/2022 • 8 minutes to read • [Edit Online](#)

Blob storage now supports the SSH File Transfer Protocol (SFTP). This support provides the ability to securely connect to Blob Storage accounts via an SFTP endpoint, allowing you to use SFTP for file access, file transfer, and file management.

## IMPORTANT

SFTP support is currently in PREVIEW and is available on general-purpose v2 and premium block blob accounts. Complete [this form](#) BEFORE using the feature in preview. Registration via 'preview features' is NOT required and confirmation email will NOT be sent after filling out the form. You can IMMEDIATELY access the feature.

After testing your end-to-end scenarios with SFTP, please share your experience via [this form](#).

See the [Supplemental Terms of Use for Microsoft Azure Previews](#) for legal terms that apply to Azure features that are in beta, preview, or otherwise not yet released into general availability.

Azure allows secure data transfer to Blob Storage accounts using Azure Blob service REST API, Azure SDKs, and tools such as AzCopy. However, legacy workloads often use traditional file transfer protocols such as SFTP. You could update custom applications to use the REST API and Azure SDKs, but only by making significant code changes.

Prior to the release of this feature, if you wanted to use SFTP to transfer data to Azure Blob Storage you would have to either purchase a third party product or orchestrate your own solution. For custom solutions, you would have to create virtual machines (VMs) in Azure to host an SFTP server, and then update, patch, manage, scale, and maintain a complex architecture.

Now, with SFTP support for Azure Blob Storage, you can enable an SFTP endpoint for Blob Storage accounts with a single click. Then you can set up local user identities for authentication to connect to your storage account with SFTP via port 22.

This article describes SFTP support for Azure Blob Storage. To learn how to enable SFTP for your storage account, see [Connect to Azure Blob Storage by using the SSH File Transfer Protocol \(SFTP\) \(preview\)](#).

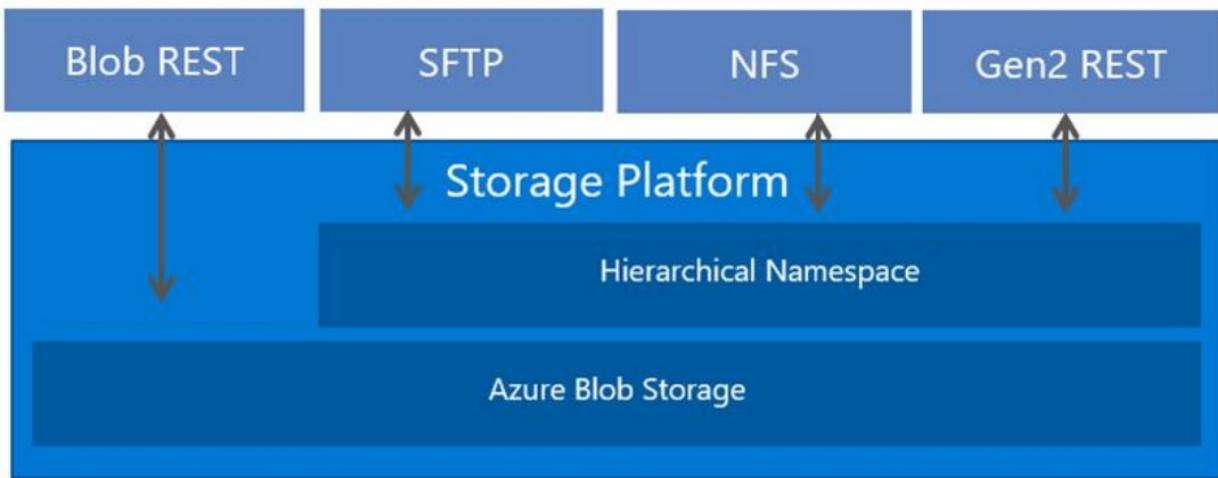
## NOTE

SFTP is a platform level service, so port 22 will be open even if the account option is disabled. If SFTP access is not configured then all requests will receive a disconnect from the service.

## SFTP and the hierarchical namespace

SFTP support requires hierarchical namespace to be enabled. Hierarchical namespace organizes objects (files) into a hierarchy of directories and subdirectories in the same way that the file system on your computer is organized. The hierarchical namespace scales linearly and doesn't degrade data capacity or performance.

Different protocols are supported by the hierarchical namespace. SFTP is one of these available protocols.



## SFTP permission model

Azure Blob Storage doesn't support Azure Active Directory (Azure AD) authentication or authorization via SFTP. Instead, SFTP utilizes a new form of identity management called *local users*.

Local users must use either a password or a Secure Shell (SSH) private key credential for authentication. You can have a maximum of 1000 local users for a storage account.

To set up access permissions, you'll create a local user, and choose authentication methods. Then, for each container in your account, you can specify the level of access you want to give that user.

### **Caution**

Local users do not interoperate with other Azure Storage permission models such as RBAC (role based access control), ABAC (attribute based access control), and ACLs (access control lists).

For example, Jeff has read only permission (can be controlled via RBAC, ABAC, or ACLs) via their Azure AD identity for file *foo.txt* stored in container *con1*. If Jeff is accessing the storage account via NFS (when not mounted as root/superuser), Blob REST, or Data Lake Storage Gen2 REST, these permissions will be enforced. However, if Jeff also has a local user identity with delete permission for data in container *con1*, they can delete *foo.txt* via SFTP using the local user identity.

For SFTP enabled storage accounts, you can use the full breadth of Azure Blob Storage security settings, to authenticate and authorize users accessing Blob Storage via Azure portal, Azure CLI, Azure PowerShell commands, AzCopy, as well as Azure SDKs, and Azure REST APIs. To learn more, see [Access control model in Azure Data Lake Storage Gen2](#).

## Authentication methods

You can authenticate local users connecting via SFTP by using a password or a Secure Shell (SSH) public-private keypair. You can configure both forms of authentication and let connecting local users choose which one to use. However, multifactor authentication, whereby both a valid password and a valid public-private key pair are required for successful authentication isn't supported.

### **Passwords**

You cannot set custom passwords, rather Azure generates one for you. If you choose password authentication, then your password will be provided after you finish configuring a local user. Make sure to copy that password and save it in a location where you can find it later. You won't be able to retrieve that password from Azure again. If you lose the password, you'll have to generate a new one. For security reasons, you can't set the password yourself.

### **SSH key pairs**

A public-private key pair is the most common form of authentication for Secure Shell (SSH). The private key is

secret and should be known only to the local user. The public key is stored in Azure. When an SSH client connects to the storage account using a local user identity, it sends a message with the private key and signature. Azure validates the message and checks that the user and key are recognized by the storage account. To learn more, see [Overview of SSH and keys](#).

If you choose to authenticate with private-public key pair, you can either generate one, use one already stored in Azure, or provide Azure the public key of an existing public-private key pair.

## Container permissions

In the current release, you can specify only container-level permissions. Directory-level permissions aren't supported. You can choose which containers you want to grant access to and what level of access you want to provide (Read, Write, List, Delete, and Create). Those permissions apply to all directories and subdirectories in the container. You can grant each local user access to as many as 100 containers. Container permissions can also be updated after creating a local user. The following table describes each permission in more detail.

PERMISSION	SYMBOL	DESCRIPTION
Read	r	<ul style="list-style-type: none"><li>• Read file content</li></ul>
Write	w	<ul style="list-style-type: none"><li>• Upload file</li><li>• Create directory</li><li>• Upload directory</li></ul>
List	l	<ul style="list-style-type: none"><li>• List content within container</li><li>• List content within directory</li></ul>
Delete	d	<ul style="list-style-type: none"><li>• Delete file/directory</li></ul>
Create	c	<ul style="list-style-type: none"><li>• Upload file if file doesn't exist</li><li>• Create directory if directory doesn't exist</li></ul>

When performing write operations on blobs in sub directories, Read permission is required to open the directory and access blob properties.

## Home directory

As you configure permissions, you have the option of setting a home directory for the local user. If no other container is specified in an SFTP connection request, then this is the directory that the user connects to by default. For example, consider the following request made by using [Open SSH](#). This request doesn't specify a container or directory name as part of the `sftp` command.

```
sftp myaccount.myusername@myaccount.blob.core.windows.net
put logfile.txt
```

If you set the home directory of a user to `mycontainer/mydirectory`, then they would connect to that directory. Then, the `logfile.txt` file would be uploaded to `mycontainer/mydirectory`. If you didn't set the home directory, then the connection attempt would fail. Instead, connecting users would have to specify a container along with the request and then use SFTP commands to navigate to the target directory before uploading a file. The following example shows this:

```
sftp myaccount.mycontainer.myusername@myaccount.blob.core.windows.net
cd mydirectory
put logfile.txt
```

#### NOTE

Home directory is only the initial directory that the connecting local user is placed in. Local users can navigate to any other path in the container they are connected to if they have the appropriate container permissions.

## Supported algorithms

You can use many different SFTP clients to securely connect and then transfer files. Connecting clients must use algorithms specified in table below.

HOST KEY <sup>1</sup>	KEY EXCHANGE	CIPHERS/ENCRYPTION	INTEGRITY/MAC	PUBLIC KEY
rsa-sha2-256 <sup>2</sup>	ecdh-sha2-nistp384	aes128-gcm@openssh.com	hmac-sha2-256	ssh-rsa <sup>2</sup>
rsa-sha2-512 <sup>2</sup>	ecdh-sha2-nistp256	aes256-gcm@openssh.com	hmac-sha2-512	ecdsa-sha2-nistp256
ecdsa-sha2-nistp256	diffie-hellman-group14-sha256	aes128-cbc	hmac-sha2-256-etm@openssh.com	ecdsa-sha2-nistp384
ecdsa-sha2-nistp384	diffie-hellman-group16-sha512	aes192-cbc	hmac-sha2-512-etm@openssh.com	
	diffie-hellman-group-exchange-sha256	aes256-cbc		
		aes128-ctr		
		aes192-ctr		
		aes256-ctr		

<sup>1</sup> Host keys are published [here](#). <sup>2</sup> RSA keys must be minimum 2048 bits in length.

SFTP support for Azure Blob Storage currently limits its cryptographic algorithm support based on security considerations. We strongly recommend that customers utilize [Microsoft Security Development Lifecycle \(SDL\) approved algorithms](#) to securely access their data.

At this time, in accordance with the Microsoft Security SDL, we do not plan on supporting the following:

`ssh-dss`, `diffie-hellman-group14-sha1`, `diffie-hellman-group1-sha1`, `hmac-sha1`, `hmac-sha1-96`. Algorithm support is subject to change in the future.

## Connecting with SFTP

To get started, enable SFTP support, create a local user, and assign permissions for that local user. Then, you can use any SFTP client to securely connect and then transfer files. For step-by-step guidance, see [Connect to Azure Blob Storage by using the SSH File Transfer Protocol \(SFTP\)](#).

## Known supported clients

The following clients have compatible algorithm support with SFTP for Azure Blob Storage (preview). See [Limitations and known issues with SSH File Transfer Protocol \(SFTP\) support for Azure Blob Storage](#) if you're having trouble connecting.

- AsyncSSH 2.1.0+
- Axway
- Cyberduck 7.8.2+
- edtFTPjPRO 7.0.0+
- FileZilla 3.53.0+
- libssh 0.9.5+
- Maverick Legacy 1.7.15+
- Moveit 12.7
- OpenSSH 7.4+
- paramiko 2.8.1+
- PuTTY 0.74+
- QualysML 12.3.41.1+
- RebexSSH 5.0.7119.0+
- Salesforce
- ssh2js 0.1.20+
- sshj 0.27.0+
- SSH.NET 2020.0.0+
- WinSCP 5.10+
- Workday
- XFB.Gateway

The supported client list above is not exhaustive and may change over time.

## Limitations and known issues

See the [limitations and known issues article](#) for a complete list of limitations and issues with SFTP support for Azure Blob Storage.

## Pricing and billing

### IMPORTANT

During the public preview, the use of SFTP does not incur any additional charges. However, the standard transaction, storage, and networking prices for the underlying Azure Data Lake Store Gen2 account still apply. SFTP might incur additional charges when the feature becomes generally available.

Transaction and storage costs are based on factors such as storage account type and the endpoint that you use to transfer data to the storage account. To learn more, see [Understand the full billing model for Azure Blob Storage](#).

## See also

- [Connect to Azure Blob Storage by using the SSH File Transfer Protocol \(SFTP\)](#)
- [Limitations and known issues with SSH File Transfer Protocol \(SFTP\) support for Azure Blob Storage](#)
- [Host keys for SSH File Transfer Protocol \(SFTP\) support for Azure Blob Storage](#)

- SSH File Transfer Protocol (SFTP) performance considerations in Azure Blob storage

# SSH File Transfer Protocol (SFTP) performance considerations in Azure Blob storage (preview)

8/22/2022 • 2 minutes to read • [Edit Online](#)

Blob storage now supports the SSH File Transfer Protocol (SFTP). This article contains recommendations that will help you to optimize the performance of your storage requests. To learn more about SFTP support for Azure Blob Storage, see [SSH File Transfer Protocol \(SFTP\) support for Azure Blob Storage](#).

## IMPORTANT

SFTP support is currently in PREVIEW and is available on general-purpose v2 and premium block blob accounts. Complete [this form](#) BEFORE using the feature in preview. Registration via 'preview features' is NOT required and confirmation email will NOT be sent after filling out the form. You can IMMEDIATELY access the feature.

After testing your end-to-end scenarios with SFTP, please share your experience via [this form](#).

See the [Supplemental Terms of Use for Microsoft Azure Previews](#) for legal terms that apply to Azure features that are in beta, preview, or otherwise not yet released into general availability.

## Use concurrent connections to increase throughput

Azure Blob Storage scales linearly until it reaches the maximum storage account egress and ingress limit. Therefore, your applications can achieve higher throughput by using more client connections. To view storage account egress and ingress limits, see [Scalability and performance targets for standard storage accounts](#).

For WinSCP, you can use a maximum of 9 concurrent connections to upload multiple files. Other common SFTP clients such as FileZilla have similar options.

## IMPORTANT

Concurrent uploads will only improve performance when uploading multiple files at the same time. Using multiple connections to upload a single file is not supported.

## Use premium block blob storage accounts

[Azure premium block blob storage account](#) offers consistent low-latency and high transaction rates. The premium block blob storage account can reach maximum bandwidth with fewer threads and clients. For example, with a single client, a premium block blob storage account can achieve 2.3x bandwidth compared to the same setup used with a standard performance general purpose v2 storage account.

## Reduce the impact of network latency

Network latency has a large impact on SFTP performance due to its reliance on small messages. By default, most clients use a message size of around 32KB.

- Increase default message size to achieve better performance
  - For OpenSSH on Windows, you can increase the message size to 100000 with the `-B` option:

```
sftp -B 100000 testaccount.user1@testaccount.blob.core.windows.net
```

- For OpenSSH on Linux, you can increase buffer size to 262000 with the `-B` option:

```
sftp -B 262000 -R 32 testaccount.user1@testaccount.blob.core.windows.net
```

- Make storage requests from a client located in the same region as the storage account

## See also

- [SSH File Transfer Protocol \(SFTP\) support for Azure Blob Storage](#)
- [Connect to Azure Blob Storage by using the SSH File Transfer Protocol \(SFTP\)](#)
- [Limitations and known issues with SSH File Transfer Protocol \(SFTP\) support for Azure Blob Storage](#)
- [Host keys for SSH File Transfer Protocol \(SFTP\) support for Azure Blob Storage](#)

# Limitations and known issues with SSH File Transfer Protocol (SFTP) support for Azure Blob Storage (preview)

8/22/2022 • 3 minutes to read • [Edit Online](#)

This article describes limitations and known issues of SFTP support for Azure Blob Storage.

## IMPORTANT

SFTP support is currently in PREVIEW and is available on general-purpose v2 and premium block blob accounts. Complete [this form](#) BEFORE using the feature in preview. Registration via 'preview features' is NOT required and confirmation email will NOT be sent after filling out the form. You can IMMEDIATELY access the feature.

After testing your end-to-end scenarios with SFTP, please share your experience via [this form](#).

See the [Supplemental Terms of Use for Microsoft Azure Previews](#) for legal terms that apply to Azure features that are in beta, preview, or otherwise not yet released into general availability.

## IMPORTANT

Because you must enable hierarchical namespace for your account to use SFTP, all of the known issues that are described in the Known issues with [Azure Data Lake Storage Gen2](#) article also apply to your account.

## Known unsupported clients

The following clients are known to be incompatible with SFTP for Azure Blob Storage (preview). See [Supported algorithms](#) for more information.

- Five9
- Kemp
- Mule
- paramiko 1.16.0
- SSH.NET 2016.1.0

The unsupported client list above is not exhaustive and may change over time.

## Unsupported operations

CATEGORY	UNSUPPORTED OPERATIONS
ACLs	<ul style="list-style-type: none"><li>• <code>chgrp</code> - change group</li><li>• <code>chmod</code> - change permissions/mode</li><li>• <code>chown</code> - change owner</li><li>• <code>put/get -p</code> - preserving permissions</li></ul>
Resume operations	<ul style="list-style-type: none"><li>• <code>reget</code> , <code>get -a</code> - resume download</li><li>• <code>reput</code> , <code>put -a</code> - resume upload</li></ul>

CATEGORY	UNSUPPORTED OPERATIONS
Random writes and appends	<ul style="list-style-type: none"> <li>Operations that include both READ and WRITE flags. For example: <a href="#">SSH.NET create API</a></li> <li>Operations that include APPEND flag. For example: <a href="#">SSH.NET append API</a>.</li> </ul>
Links	<ul style="list-style-type: none"> <li><code>symlink</code> - creating symbolic links</li> <li><code>ln</code> - creating hard links</li> <li>Reading links not supported</li> </ul>
Capacity Information	<code>df</code> - usage info for filesystem
Extensions	Unsupported extensions include but aren't limited to: <code>fsync@openssh.com</code> , <code>limits@openssh.com</code> , <code>lsetstat@openssh.com</code> , <code>statvfs@openssh.com</code>
SSH Commands	SFTP is the only supported subsystem. Shell requests after the completion of key exchange will fail.
Multi-protocol writes	Random writes and appends ( <code>PutBlock</code> , <code>PutBlockList</code> , <code>GetBlockList</code> , <code>AppendBlock</code> , <code>AppendFile</code> ) aren't allowed from other protocols (NFS, Blob REST, Data Lake Storage Gen2 REST) on blobs that are created by using SFTP. Full overwrites are allowed.

## Authentication and authorization

- Local users* is the only form of identity management that is currently supported for the SFTP endpoint.
- Azure Active Directory (Azure AD) isn't supported for the SFTP endpoint.
- POSIX-like access control lists (ACLs) aren't supported for the SFTP endpoint.

To learn more, see [SFTP permission model](#) and see [Access control model in Azure Data Lake Storage Gen2](#).

## Networking

- To access the storage account using SFTP, your network must allow traffic on port 22.
- Static IP addresses aren't supported for storage accounts. This is not an SFTP specific limitation.
- Internet routing is not supported. Use Microsoft network routing.
- There's a 2 minute timeout for idle or inactive connections. OpenSSH will appear to stop responding and then disconnect. Some clients reconnect automatically.

## Other

- For performance issues and considerations, see [SSH File Transfer Protocol \(SFTP\) performance considerations in Azure Blob storage](#).
- Special containers such as \$logs, \$blobchangefeed, \$root, \$web aren't accessible via the SFTP endpoint.
- Symbolic links aren't supported.
- SSH and SCP commands that aren't SFTP aren't supported.

- FTPS and FTP aren't supported.
- TLS and SSL aren't related to SFTP.

## Troubleshooting

- To resolve the

```
Failed to update SFTP settings for account 'accountname'. Error: The value 'True' is not allowed for property isSftpEnabled.
```

error, ensure that the following pre-requisites are met at the storage account level:

- The account needs to be a general-purpose v2 and premium block blob accounts.
- The account needs to have hierarchical namespace enabled on it.

- To resolve the `Home Directory not accessible` error, check that:

- The user has been assigned appropriate permissions to the container.
- The container name is specified in the connection string for local users don't have a home directory.
- The container name is specified in the connection string for local users that have a home directory that doesn't exist.

- To resolve the `Received disconnect from XX.XXX.XX.XXX port 22:11:` when connecting, check that:

- Public network access is `Enabled from all networks` or  
`Enabled from selected virtual networks and IP addresses`.
- The client IP address is allowed by the firewall.
- Network Routing is set to `Microsoft network routing`.

## See also

- [SSH File Transfer Protocol \(SFTP\) support for Azure Blob Storage](#)
- [Connect to Azure Blob Storage by using the SSH File Transfer Protocol \(SFTP\)](#)
- [Host keys for SSH File Transfer Protocol \(SFTP\) support for Azure Blob Storage](#)
- [SSH File Transfer Protocol \(SFTP\) performance considerations in Azure Blob storage](#)

# Reacting to Blob storage events

8/22/2022 • 4 minutes to read • [Edit Online](#)

Azure Storage events allow applications to react to events, such as the creation and deletion of blobs. It does so without the need for complicated code or expensive and inefficient polling services. The best part is you only pay for what you use.

Blob storage events are pushed using [Azure Event Grid](#) to subscribers such as Azure Functions, Azure Logic Apps, or even to your own http listener. Event Grid provides reliable event delivery to your applications through rich retry policies and dead-lettering.

See the [Blob storage events schema](#) article to view the full list of the events that Blob storage supports.

Common Blob storage event scenarios include image or video processing, search indexing, or any file-oriented workflow. Asynchronous file uploads are a great fit for events. When changes are infrequent, but your scenario requires immediate responsiveness, event-based architecture can be especially efficient.

If you want to try blob storage events, see any of these quickstart articles:

IF YOU WANT TO USE THIS TOOL:	SEE THIS ARTICLE:
Azure portal	<a href="#">Quickstart: Route Blob storage events to web endpoint with the Azure portal</a>
PowerShell	<a href="#">Quickstart: Route storage events to web endpoint with PowerShell</a>
Azure CLI	<a href="#">Quickstart: Route storage events to web endpoint with Azure CLI</a>

To view in-depth examples of reacting to Blob storage events by using Azure functions, see these articles:

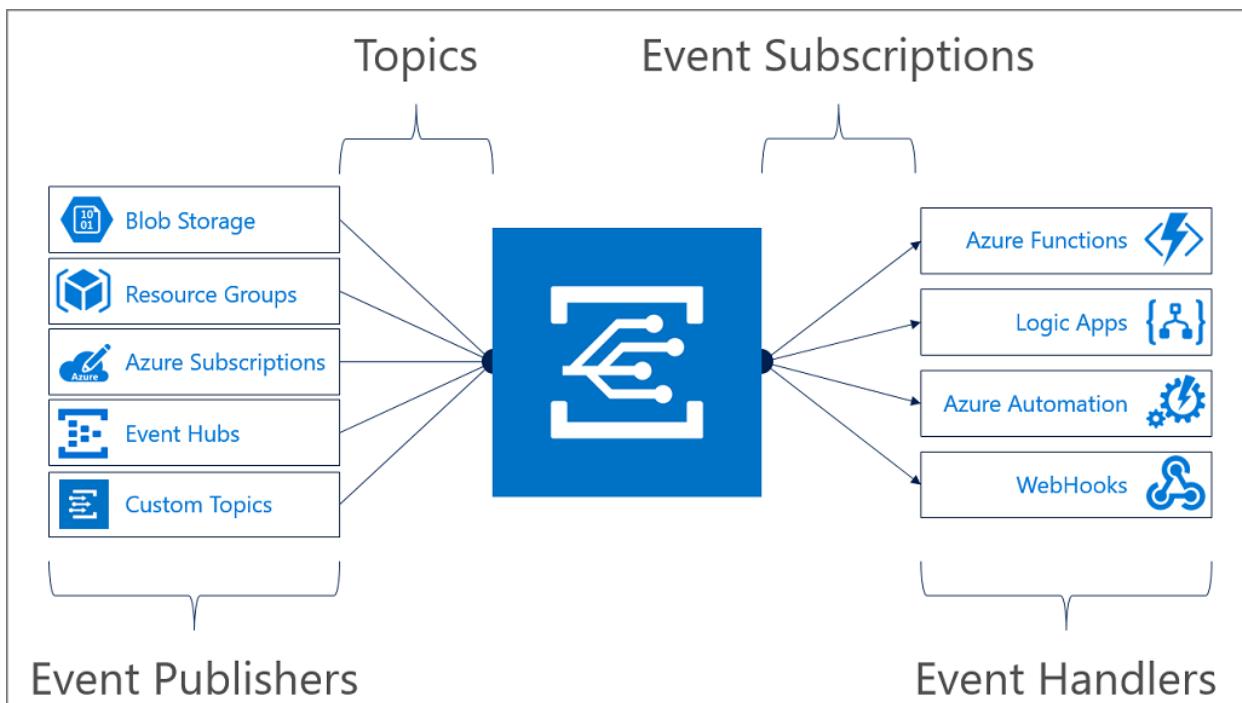
- [Tutorial: Use Azure Data Lake Storage Gen2 events to update a Databricks Delta table](#).
- [Tutorial: Automate resizing uploaded images using Event Grid](#)

## NOTE

Storage (general purpose v1) does *not* support integration with Event Grid.

## The event model

Event Grid uses [event subscriptions](#) to route event messages to subscribers. This image illustrates the relationship between event publishers, event subscriptions, and event handlers.



First, subscribe an endpoint to an event. Then, when an event is triggered, the Event Grid service will send data about that event to the endpoint.

See the [Blob storage events schema](#) article to view:

- A complete list of Blob storage events and how each event is triggered.
- An example of the data the Event Grid would send for each of these events.
- The purpose of each key value pair that appears in the data.

## Filtering events

Blob [events can be filtered](#) by the event type, container name, or name of the object that was created/deleted. Filters in Event Grid match the beginning or end of the subject so events with a matching subject go to the subscriber.

To learn more about how to apply filters, see [Filter events for Event Grid](#).

The subject of Blob storage events uses the format:

```
/blobServices/default/containers/<containername>/blobs/<blobname>
```

To match all events for a storage account, you can leave the subject filters empty.

To match events from blobs created in a set of containers sharing a prefix, use a `subjectBeginsWith` filter like:

```
/blobServices/default/containers/containerprefix
```

To match events from blobs created in specific container, use a `subjectBeginsWith` filter like:

```
/blobServices/default/containers/containername/
```

To match events from blobs created in specific container sharing a blob name prefix, use a `subjectBeginsWith` filter like:

```
/blobServices/default/containers/containername/blobs/blobprefix
```

To match events from blobs created in specific container sharing a blob suffix, use a `subjectEndsWith` filter like `".log"` or `".jpg"`. For more information, see [Event Grid Concepts](#).

## Practices for consuming events

Applications that handle Blob storage events should follow a few recommended practices:

- As multiple subscriptions can be configured to route events to the same event handler, it is important not to assume events are from a particular source, but to check the topic of the message to ensure that it comes from the storage account you are expecting.
- Similarly, check that the `eventType` is one you are prepared to process, and do not assume that all events you receive will be the types you expect.
- As messages can arrive after some delay, use the `etag` fields to understand if your information about objects is still up-to-date. To learn how to use the `etag` field, see [Managing concurrency in Blob storage](#).
- As messages can arrive out of order, use the `sequencer` fields to understand the order of events on any particular object. The `sequencer` field is a string value that represents the logical sequence of events for any particular blob name. You can use standard string comparison to understand the relative sequence of two events on the same blob name.
- Storage events guarantees at-least-once delivery to subscribers, which ensures that all messages are outputted. However due to retries between backend nodes and services or availability of subscriptions, duplicate messages may occur. To learn more about message delivery and retry, see [Event Grid message delivery and retry](#).
- Use the `blobType` field to understand what type of operations are allowed on the blob, and which client library types you should use to access the blob. Valid values are either `BlockBlob` or `PageBlob`.
- Use the `url` field with the `CloudBlockBlob` and `CloudAppendBlob` constructors to access the blob.
- Ignore fields you don't understand. This practice will help keep you resilient to new features that might be added in the future.
- If you want to ensure that the `Microsoft.Storage.BlobCreated` event is triggered only when a Block Blob is completely committed, filter the event for the `CopyBlob`, `PutBlob`, `PutBlockList` or `FlushWithClose` REST API calls. These API calls trigger the `Microsoft.Storage.BlobCreated` event only after data is fully committed to a Block Blob. To learn how to create a filter, see [Filter events for Event Grid](#).

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## Next steps

Learn more about Event Grid and give Blob storage events a try:

- [About Event Grid](#)
- [Blob storage events schema](#)
- [Route Blob storage Events to a custom web endpoint](#)

# Overview of Azure page blobs

8/22/2022 • 10 minutes to read • [Edit Online](#)

Azure Storage offers three types of blob storage: Block Blobs, Append Blobs and page blobs. Block blobs are composed of blocks and are ideal for storing text or binary files, and for uploading large files efficiently. Append blobs are also made up of blocks, but they are optimized for append operations, making them ideal for logging scenarios. Page blobs are made up of 512-byte pages up to 8 TB in total size and are designed for frequent random read/write operations. Page blobs are the foundation of Azure IaaS Disks. This article focuses on explaining the features and benefits of page blobs.

Page blobs are a collection of 512-byte pages, which provide the ability to read/write arbitrary ranges of bytes. Hence, page blobs are ideal for storing index-based and sparse data structures like OS and data disks for Virtual Machines and Databases. For example, Azure SQL DB uses page blobs as the underlying persistent storage for its databases. Moreover, page blobs are also often used for files with Range-Based updates.

Key features of Azure page blobs are its REST interface, the durability of the underlying storage, and the seamless migration capabilities to Azure. These features are discussed in more detail in the next section. In addition, Azure page blobs are currently supported on two types of storage: Premium Storage and Standard Storage. Premium Storage is designed specifically for workloads requiring consistent high performance and low latency making premium page blobs ideal for high performance storage scenarios. Standard storage accounts are more cost effective for running latency-insensitive workloads.

## Restrictions

Page blobs can only use the **Hot** access tier, they cannot use either the **Cool** or **Archive** tiers. For more information on access tiers, see [Hot, Cool, and Archive access tiers for blob data](#).

## Sample use cases

Let's discuss a couple of use cases for page blobs starting with Azure IaaS Disks. Azure page blobs are the backbone of the virtual disks platform for Azure IaaS. Both Azure OS and data disks are implemented as virtual disks where data is durably persisted in the Azure Storage platform and then delivered to the virtual machines for maximum performance. Azure Disks are persisted in Hyper-V [VHD format](#) and stored as a [page blob](#) in Azure Storage. In addition to using virtual disks for Azure IaaS VMs, page blobs also enable PaaS and DBaaS scenarios such as Azure SQL DB service, which currently uses page blobs for storing SQL data, enabling fast random read-write operations for the database. Another example would be if you have a PaaS service for shared media access for collaborative video editing applications, page blobs enable fast access to random locations in the media. It also enables fast and efficient editing and merging of the same media by multiple users.

First party Microsoft services like Azure Site Recovery, Azure Backup, as well as many third-party developers have implemented industry-leading innovations using page blob's REST interface. Following are some of the unique scenarios implemented on Azure:

- Application-directed incremental snapshot management: Applications can leverage page blob snapshots and REST APIs for saving the application checkpoints without incurring costly duplication of data. Azure Storage supports local snapshots for page blobs, which don't require copying the entire blob. These public snapshot APIs also enable accessing and copying of deltas between snapshots.
- Live migration of application and data from on premises to cloud: Copy the on premises data and use REST APIs to write directly to an Azure page blob while the on premises VM continues to run. Once the target has caught up, you can quickly failover to Azure VM using that data. In this way, you can migrate your VMs and

virtual disks from on premises to cloud with minimal downtime since the data migration occurs in the background while you continue to use the VM and the downtime needed for failover will be short (in minutes).

- [SAS-based](#) shared access, which enables scenarios like multiple-readers and single-writer with support for concurrency control.

## Pricing

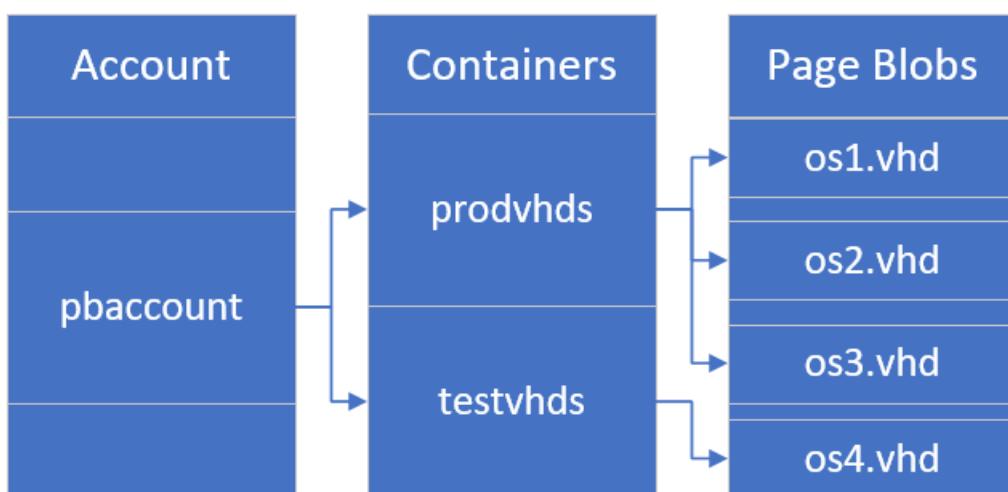
Both types of storage offered with page blobs have their own pricing model. Premium page blobs follow the managed disks pricing model, while standard page blobs are billed on used size and with each transaction. For more information, see the [Azure Page Blobs pricing page](#).

## Page blob features

### REST API

Refer to the following document to get started with [developing using page blobs](#). As an example, look at how to access page blobs using Storage Client Library for .NET.

The following diagram describes the overall relationships between account, containers, and page blobs.



### Creating an empty page blob of a specified size

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

First, get a reference to a container. To create a page blob, call the `GetPageBlobClient` method, and then call the `PageBlobClient.Create` method. Pass in the max size for the blob to create. That size must be a multiple of 512 bytes.

```
long OneGigabyteAsBytes = 1024 * 1024 * 1024;

BlobServiceClient blobServiceClient = new BlobServiceClient(connectionString);

var blobContainerClient =
 blobServiceClient.GetBlobContainerClient(Constants.containerName);

var pageBlobClient = blobContainerClient.GetPageBlobClient("0s4.vhd");

pageBlobClient.Create(16 * OneGigabyteAsBytes);
```

### Resizing a page blob

- [.NET v12 SDK](#)

- .NET v11 SDK

To resize a page blob after creation, use the [Resize](#) method. The requested size should be a multiple of 512 bytes.

```
pageBlobClient.Resize(32 * OneGigabyteAsBytes);
```

#### Writing pages to a page blob

- .NET v12 SDK
- .NET v11 SDK

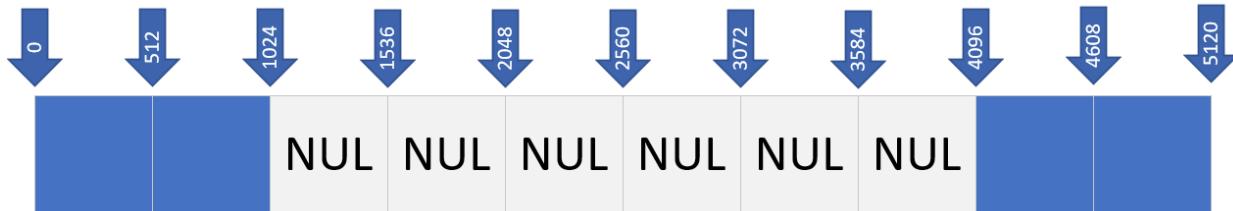
To write pages, use the [PageBlobClient.UploadPages](#) method.

```
pageBlobClient.UploadPages(dataStream, startingOffset);
```

This allows you to write a sequential set of pages up to 4MBs. The offset being written to must start on a 512-byte boundary ( $startingOffset \% 512 == 0$ ), and end on a 512 boundary - 1.

As soon as a write request for a sequential set of pages succeeds in the blob service and is replicated for durability and resiliency, the write has committed, and success is returned back to the client.

The below diagram shows 2 separate write operations:



1. A Write operation starting at offset 0 of length 1024 bytes
2. A Write operation starting at offset 4096 of length 1024

#### Reading pages from a page blob

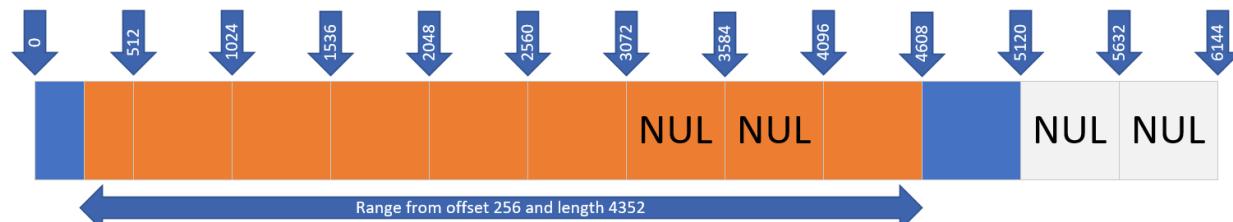
- .NET v12 SDK
- .NET v11 SDK

To read pages, use the [PageBlobClient.Download](#) method to read a range of bytes from the page blob.

```
var pageBlob = pageBlobClient.Download(new HttpRange(bufferOffset, rangeSize));
```

This allows you to download the full blob or range of bytes starting from any offset in the blob. When reading, the offset does not have to start on a multiple of 512. When reading bytes from a NUL page, the service returns zero bytes.

The following figure shows a Read operation with an offset of 256 and a range size of 4352. Data returned is highlighted in orange. Zeros are returned for NUL pages.



If you have a sparsely populated blob, you may want to just download the valid page regions to avoid paying for egressing of zero bytes and to reduce download latency.

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

To determine which pages are backed by data, use `PageBlobClient.GetPageRanges`. You can then enumerate the returned ranges and download the data in each range.

```
IEnumerable<HttpRange> pageRanges = pageBlobClient.GetPageRanges().Value.PageRanges;

foreach (var range in pageRanges)
{
 var pageBlob = pageBlobClient.Download(range);
}
```

### **Leasing a page blob**

The Lease Blob operation establishes and manages a lock on a blob for write and delete operations. This operation is useful in scenarios where a page blob is being accessed from multiple clients to ensure only one client can write to the blob at a time. Azure Disks, for example, leverages this leasing mechanism to ensure the disk is only managed by a single VM. The lock duration can be 15 to 60 seconds, or can be infinite. See the documentation [here](#) for more details.

In addition to rich REST APIs, page blobs also provide shared access, durability, and enhanced security. We will cover those benefits in more detail in the next paragraphs.

### **Concurrent access**

The page blobs REST API and its leasing mechanism allows applications to access the page blob from multiple clients. For example, let's say you need to build a distributed cloud service that shares storage objects with multiple users. It could be a web application serving a large collection of images to several users. One option for implementing this is to use a VM with attached disks. Downsides of this include, (i) the constraint that a disk can only be attached to a single VM thus limiting the scalability, flexibility, and increasing risks. If there is a problem with the VM or the service running on the VM, then due to the lease, the image is inaccessible until the lease expires or is broken; and (ii) Additional cost of having an IaaS VM.

An alternative option is to use the page blobs directly via Azure Storage REST APIs. This option eliminates the need for costly IaaS VMs, offers full flexibility of direct access from multiple clients, simplifies the classic deployment model by eliminating the need to attach/detach disks, and eliminates the risk of issues on the VM. And, it provides the same level of performance for random read/write operations as a disk

### **Durability and high availability**

Both Standard and premium storage are durable storage where the page blob data is always replicated to ensure durability and high availability. For more information about Azure Storage Redundancy, see this [documentation](#). Azure has consistently delivered enterprise-grade durability for IaaS disks and page blobs, with an industry-leading zero percent [Annualized Failure Rate](#).

### **Seamless migration to Azure**

For the customers and developers who are interested in implementing their own customized backup solution, Azure also offers incremental snapshots that only hold the deltas. This feature avoids the cost of the initial full copy, which greatly lowers the backup cost. Along with the ability to efficiently read and copy differential data, this is another powerful capability that enables even more innovations from developers, leading to a best-in-class backup and disaster recovery (DR) experience on Azure. You can set up your own backup or DR solution for your VMs on Azure using [Blob Snapshot](#) along with the [Get Page Ranges API](#) and the [Incremental Copy Blob API](#), which you can use for easily copying the incremental data for DR.

Moreover, many enterprises have critical workloads already running in on-premises datacenters. For migrating the workload to the cloud, one of the main concerns would be the amount of downtime needed for copying the data, and the risk of unforeseen issues after the switchover. In many cases, the downtime can be a showstopper for migration to the cloud. Using the page blobs REST API, Azure addresses this problem by enabling cloud migration with minimal disruption to critical workloads.

For examples on how to take a snapshot and how to restore a page blob from a snapshot, please refer to the [setup a backup process using incremental snapshots](#) article.

# Static website hosting in Azure Storage

8/22/2022 • 7 minutes to read • [Edit Online](#)

You can serve static content (HTML, CSS, JavaScript, and image files) directly from a storage container named `$web`. Hosting your content in Azure Storage enables you to use serverless architectures that include [Azure Functions](#) and other Platform as a service (PaaS) services. Azure Storage static website hosting is a great option in cases where you don't require a web server to render content.

Static websites have some limitations. For example, If you want to configure headers, you'll have to use Azure Content Delivery Network (Azure CDN). There's no way to configure headers as part of the static website feature itself. Also, AuthN and AuthZ are not supported.

If these features are important for your scenario, consider using [Azure Static Web Apps](#). It's a great alternative to static websites and is also appropriate in cases where you don't require a web server to render content. You can configure headers and AuthN / AuthZ is fully supported. Azure Static Web Apps also provides a fully managed continuous integration and continuous delivery (CI/CD) workflow from GitHub source to global deployment.

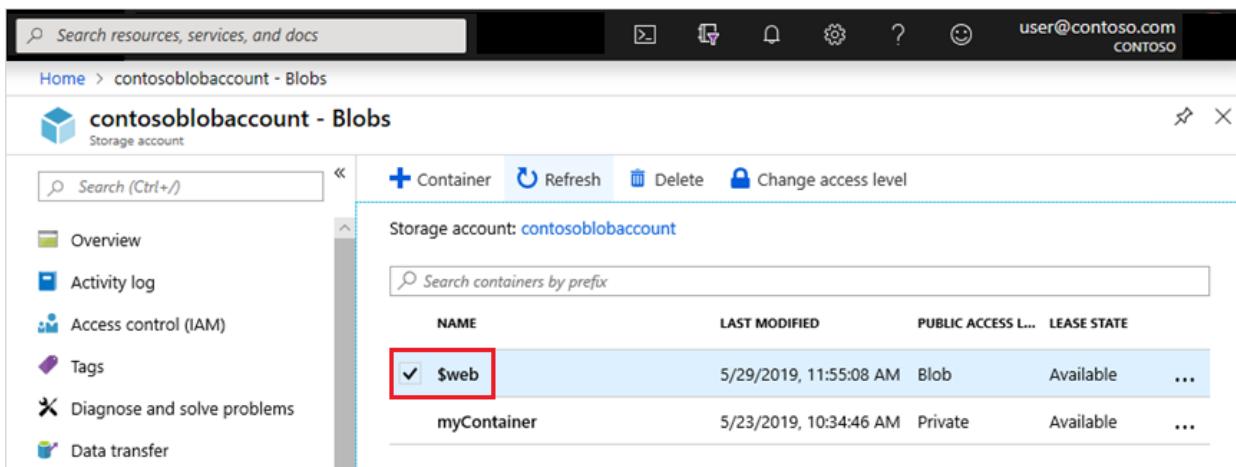
If you need a web server to render content, you can use [Azure App Service](#).

## Setting up a static website

Static website hosting is a feature that you have to enable on the storage account.

To enable static website hosting, select the name of your default file, and then optionally provide a path to a custom 404 page. If a blob storage container named `$web` doesn't already exist in the account, one is created for you. Add the files of your site to this container.

For step-by-step guidance, see [Host a static website in Azure Storage](#).



The screenshot shows the Azure Storage Blobs blade for the 'contosoblobaccount' storage account. On the left, there's a navigation menu with options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, and Data transfer. The main area displays a table of blobs. The first row, '\$web', is highlighted with a red box and has a checkmark icon next to it. The table columns are NAME, LAST MODIFIED, PUBLIC ACCESS L..., and LEASE STATE. The '\$web' row shows '5/29/2019, 11:55:08 AM', 'Blob', 'Available', and an ellipsis (...). The second row, 'myContainer', shows '5/23/2019, 10:34:46 AM', 'Private', 'Available', and an ellipsis (...). At the top of the blade, there are buttons for Container, Refresh, Delete, and Change access level, along with a search bar for containers by prefix.

NAME	LAST MODIFIED	PUBLIC ACCESS L...	LEASE STATE
\$web	5/29/2019, 11:55:08 AM	Blob	Available
myContainer	5/23/2019, 10:34:46 AM	Private	Available

Files in the `$web` container are case-sensitive, served through anonymous access requests and are available only through read operations.

## Uploading content

You can use any of these tools to upload content to the `$web` container:

- [Azure CLI](#)
- [Azure PowerShell module](#)
- [AzCopy](#)

- [Azure Storage Explorer](#)
- [Azure Pipelines](#)
- [Visual Studio Code extension](#) and [Channel 9 video demonstration](#)

## Viewing content

Users can view site content from a browser by using the public URL of the website. You can find the URL by using the Azure portal, Azure CLI, or PowerShell. See [Find the website URL](#).

The index document that you specify when you enable static website hosting appears when users open the site and don't specify a specific file (For example: `https://contosoblobaccount.z22.web.core.windows.net` ).

If the server returns a 404 error, and you have not specified an error document when you enabled the website, then a default 404 page is returned to the user.

### NOTE

Cross-Origin Resource Sharing (CORS) support for Azure Storage is not supported with static website.

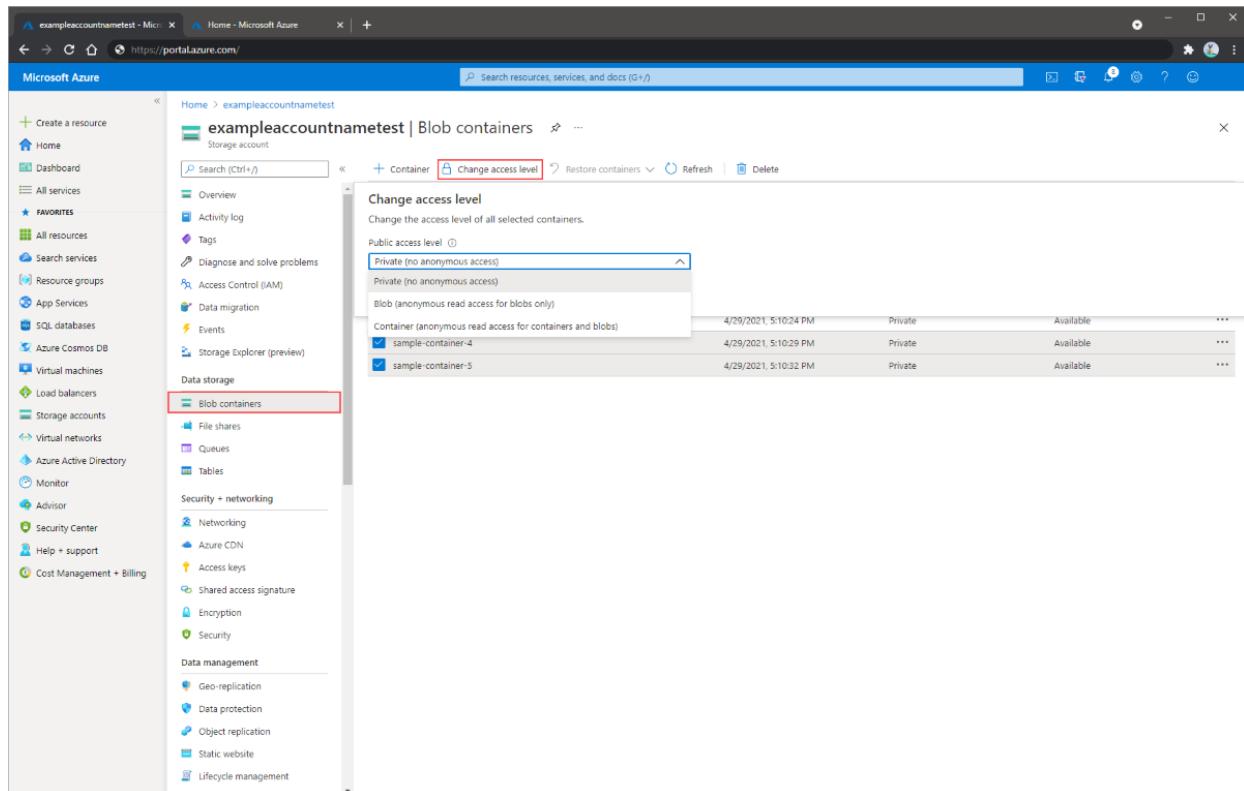
## Secondary endpoints

If you set up [redundancy in a secondary region](#), you can also access website content by using a secondary endpoint. Because data is replicated to secondary regions asynchronously, the files that are available at the secondary endpoint aren't always in sync with the files that are available on the primary endpoint.

## Impact of the setting the public access level of the web container

You can modify the public access level of the `$web` container, but this has no impact on the primary static website endpoint because these files are served through anonymous access requests. That means public (read-only) access to all files.

The following screenshot shows the public access level setting in the Azure portal:



While the primary static website endpoint is not affected, a change to the public access level does impact the

primary blob service endpoint.

For example, if you change the public access level of the `$web` container from **Private (no anonymous access)** to **Blob (anonymous read access for blobs only)**, then the level of public access to the primary static website endpoint `https://contosoblobaccount.z22.web.core.windows.net/index.html` doesn't change.

However, the public access to the primary blob service endpoint

`https://contosoblobaccount.blob.core.windows.net/$web/index.html` does change from private to public. Now users can open that file by using either of these two endpoints.

Disabling public access on a storage account does not affect static websites that are hosted in that storage account. For more information, see [Configure anonymous public read access for containers and blobs](#).

## Mapping a custom domain to a static website URL

You can make your static website available via a custom domain.

It's easier to enable HTTP access for your custom domain, because Azure Storage natively supports it. To enable HTTPS, you'll have to use Azure CDN because Azure Storage does not yet natively support HTTPS with custom domains. see [Map a custom domain to an Azure Blob Storage endpoint](#) for step-by-step guidance.

If the storage account is configured to [require secure transfer](#) over HTTPS, then users must use the HTTPS endpoint.

### TIP

Consider hosting your domain on Azure. For more information, see [Host your domain in Azure DNS](#).

## Adding HTTP headers

There's no way to configure headers as part of the static website feature. However, you can use Azure CDN to add headers and append (or overwrite) header values. See [Standard rules engine reference for Azure CDN](#).

If you want to use headers to control caching, see [Control Azure CDN caching behavior with caching rules](#).

## Multi-region website hosting

If you plan to host a website in multiple geographies, we recommend that you use a [Content Delivery Network](#) for regional caching. Use [Azure Front Door](#) if you want to serve different content in each region. It also provides failover capabilities. [Azure Traffic Manager](#) is not recommended if you plan to use a custom domain. Issues can arise because of how Azure Storage verifies custom domain names.

## Permissions

The permission to be able to enable static website is `Microsoft.Storage/storageAccounts/blobServices/write` or shared key. Built in roles that provide this access include Storage Account Contributor.

## Pricing

You can enable static website hosting free of charge. You're billed only for the blob storage that your site utilizes and operations costs. For more details on prices for Azure Blob Storage, check out the [Azure Blob Storage Pricing Page](#).

## Metrics

You can enable metrics on static website pages. Once you've enabled metrics, traffic statistics on files in the **\$web** container are reported in the metrics dashboard.

To enable metrics on your static website pages, see [Enable metrics on static website pages](#).

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## FAQ

**Does the Azure Storage firewall work with a static website?**

Yes. Storage account [network security rules](#), including IP-based and VNET firewalls, are supported for the static website endpoint, and may be used to protect your website.

**Do static websites support Azure Active Directory (Azure AD)?**

No. A static website only supports anonymous public read access for files in the **\$web** container.

**How do I use a custom domain with a static website?**

You can configure a [custom domain](#) with a static website by using [Azure Content Delivery Network \(Azure CDN\)](#). Azure CDN provides consistent low latencies to your website from anywhere in the world.

**How do I use a custom SSL certificate with a static website?**

You can configure a [custom SSL](#) certificate with a static website by using [Azure CDN](#). Azure CDN provides consistent low latencies to your website from anywhere in the world.

**How do I add custom headers and rules with a static website?**

You can configure the host header for a static website by using [Azure CDN - Verizon Premium](#). We'd be interested to hear your feedback [here](#).

**Why am I getting an HTTP 404 error from a static website?**

This can happen if you refer to a file name by using an incorrect case. For example: `Index.html` instead of `index.html`. File names and extensions in the url of a static website are case-sensitive even though they're served over HTTP. This can also happen if your Azure CDN endpoint is not yet provisioned. Wait up to 90 minutes after you provision a new Azure CDN for the propagation to complete.

**Why isn't the root directory of the website not redirecting to the default index page?**

In the Azure portal, open the static website configuration page of your account and locate the name and extension that is set in the **Index document name** field. Ensure that this name is exactly the same as the name of the file located in the **\$web** container of the storage account. File names and extensions in the url of a static website are case-sensitive even though they're served over HTTP.

## Next steps

- [Host a static website in Azure Storage](#)
- [Map a custom domain to an Azure Blob Storage endpoint](#)
- [Azure Functions](#)
- [Azure App Service](#)
- [Build your first serverless web app](#)
- [Tutorial: Host your domain in Azure DNS](#)

# Upgrading Azure Blob Storage with Azure Data Lake Storage Gen2 capabilities

8/22/2022 • 8 minutes to read • [Edit Online](#)

This article helps you to enable a hierarchical namespace and unlock capabilities such as file and directory-level security and faster operations. These capabilities are widely used by big data analytics workloads and are referred to collectively as Azure Data Lake Storage Gen2. The most popular capabilities include:

- Higher throughput, input/output operations per second (IOPS), and storage capacity limits.
- Faster operations (such as rename operations) because you can operate on individual node URLs.
- Efficient query engine that transfers only the data required to perform a given operation.
- Security at the container, directory, and file-level.

To learn more about them, see [Introduction to Azure Data Lake storage Gen2](#).

This article helps you evaluate the impact on workloads, applications, costs, service integrations, tools, features, and documentation. Make sure to review these impacts carefully. When you are ready to upgrade an account, see this step-by-step guide: [Upgrade Azure Blob Storage with Azure Data Lake Storage Gen2 capabilities](#).

## IMPORTANT

An upgrade is one-way. There's no way to revert your account once you've performed the upgrade. We recommend that you validate your upgrade in a nonproduction environment.

## Impact on availability

Make sure to plan for some downtime in your account while the upgrade process completes. Write operations are disabled while your account is being upgraded. Read operations aren't disabled, but we strongly recommend that you suspend read operations as those operations might destabilize the upgrade process.

## Impact on workloads and applications

Blob APIs work with accounts that have a hierarchical namespace, so most applications that interact with your account by using these APIs continue to work without modification.

For a complete list of issues and workarounds, see [Known issues with Blob Storage APIs](#).

Any Hadoop workloads that use Windows Azure Storage Blob driver or [WASB](#) driver, must be modified to use the [Azure Blob File System \(ABFS\)](#) driver. Unlike the WASB driver that makes requests to the **Blob service** endpoint, the ABFS driver will make requests to the **Data Lake Storage** endpoint of your account.

### Data Lake Storage endpoint

Your upgraded account will have a Data Lake storage endpoint. You can find the URL of this endpoint in the Azure portal by opening the **Properties** page of your account.

The screenshot shows the Microsoft Azure Storage account 'contoso' Endpoints page. On the left, there's a sidebar with options like Settings, Configuration, Resource sharing (CORS), Advisor recommendations, Endpoints (which is selected and highlighted in blue), and Locks. The main area has two sections: 'Blob service' and 'Data Lake Storage'. Under 'Blob service', there's a Resource ID and a URL. Under 'Data Lake Storage', there's also a Resource ID and a URL. The URL for 'Data Lake Storage' is <https://normestastorageaccount.dfs.core.windows.net/>, which is highlighted with a red box.

You don't have to modify your existing applications and workloads to use that endpoint. [Multiprotocol access in Data Lake Storage](#) makes it possible for you to use either the Blob service endpoint or the Data Lake storage endpoint to interact with your data.

Azure services and tools (such as AzCopy) might use the Data Lake storage endpoint to interact with the data in your storage account. Also, you'll need to use this new endpoint for any operations that you perform by using the [Data Lake Storage Gen2 SDKs](#), [PowerShell commands](#), or [Azure CLI commands](#).

## Directories

A Blob storage account that does not have a hierarchical namespace organizes files in a flat paradigm, rather than a hierarchical paradigm. Blobs are organized into virtual directories in order to mimic a folder structure. A virtual directory forms part of the name of the blob and is indicated by the delimiter character. Because a virtual directory is a part of the blob name, it doesn't actually exist as an independent object.

Your new account has a hierarchical namespace. That means that directories are not virtual. They are concrete, independent objects that you can operate on directly. A directory can exist without containing any files. When you delete a directory, all of the files in that directory are removed. You no longer have to delete each individual blob before the directory disappears.

## Blob metadata

Before migration, blob metadata is associated with the blob name along with its entire virtual path. After migration, the metadata is associated only with the blob. The virtual path to the blob becomes a collection of directories. Metadata of a blob is not applied to any of those directories.

## Put operations

When you upload a blob, and the path that you specify includes a directory that doesn't exist, the operation creates that directory, and then adds a blob to it. This behavior is logical in the context of a hierarchical folder structure. In a Blob storage account that does not have a hierarchical namespace, the operation doesn't create a directory. Instead, the directory name is added to the blob's namespace.

## List operations

A [List Blobs](#) operation returns both directories and files. Each is listed separately. Directories appear in the list as zero-length blobs. In a Blob storage account that does not have a hierarchical namespace, a [List Blobs](#) operation returns only blobs and not directories. If you use the Data Lake Storage Gen2 [Path - List](#) operation, directories will appear as directory entries and not as zero-length blobs.

The list order is different as well. Directories and files appear in *depth-first search* order. A Blob storage account that does not have a hierarchical namespace lists blobs in *lexicographical* order.

## Operations to rename blobs

Renaming a blob is far more efficient because client applications can rename a blob in a single operation. In

accounts that *do not* have a hierarchical namespace, tools and applications have to copy a blob and then delete the source blob.

#### NOTE

When you rename a blob, the last modified time of the blob is not updated. That's because the contents of the blob are unchanged.

## Impact on costs

Storage costs aren't impacted, but transactions costs are impacted. Use these pages to assess compare costs.

- [Block blob pricing](#).
- [Azure Data Lake Storage Gen2 pricing](#).

You can also use the **Storage Accounts** option in the [Azure Pricing Calculator](#) to estimate the impact of costs after an upgrade.

Aside from pricing changes, consider the costs savings associated with Data Lake Storage Gen2 capabilities. Overall total of cost of ownership typically declines because of higher throughput and optimized operations. Higher throughput enables you to transfer more data in less time. A hierarchical namespace improves the efficiency of operations.

## Impact on service integrations

While most Azure service integrations will continue to work after you've enable these capabilities, some of them remain in preview or not yet supported. See [Azure services that support Azure Data Lake Storage Gen2](#) to understand the current support for Azure service integrations with Data Lake Storage Gen2.

## Impact on tools, features and documentation

After you upgrade, the way that interact with some features will change. This section describes those changes.

### Blob Storage feature support

While most of Blob storage features will continue to work after you've enable these capabilities, some of them remain in preview or not yet supported.

See [Blob Storage features available in Azure Data Lake Storage Gen2](#) to understand the current support for Blob storage features with Data Lake Storage Gen2.

### Diagnostic logs

If you enable [Storage analytics logging](#), you now have the option to use the version 2.0 log format.

You don't have to use this new version. However, any operations that are applied to the Data Lake storage endpoint are recorded only in version 2.0 logs. Some services and tools that you use (such as AzCopy) will use that endpoint to perform operations on your account. To ensure that you capture logging information from all activity, consider using the version 2.0 log format.

### Azure Lifecycle management

Policies that move or delete all of the blobs in a directory won't delete the directory that contains those blobs until the next day. That's because the directory can't be deleted until all of the blobs that are located in that directory are first removed. The next day, the directory will be removed.

### Event Grid

Your new account has two endpoints: the Data Lake storage endpoint, and the Blob service endpoint. Services,

tools, and applications can use either endpoint to operate on your data. As a result, an event response that is returned by the Event Grid can show either of these two endpoints in the **url** field that describes the affected blob.

The following JSON shows the url of a blob that appears in the event response when a blob is created by using the Blob service endpoint.

```
{
 "topic": "/subscriptions/{subscription-
id}/resourceGroups/Storage/providers/Microsoft.Storage/storageAccounts/my-storage-account",
 "subject": "/blobServices/default/containers/test-container/blobs/new-file.txt",
 "eventType": "Microsoft.Storage.BlobCreated",
 "eventTime": "2017-06-26T18:41:00.9584103Z",
 "id": "831e1650-001e-001b-66ab-eeb76e069631",
 "data": {
 "api": "PutBlockList",
 "clientRequestId": "6d79dbfb-0e37-4fc4-981f-442c9ca65760",
 "requestId": "831e1650-001e-001b-66ab-eeb76e000000",
 "eTag": "\\"0x8D4BCC2E4835CD0\\\"",
 "contentType": "text/plain",
 "contentLength": 524288,
 "blobType": "BlockBlob",
 "url": "https://my-storage-account.blob.core.windows.net/testcontainer/new-file.txt",
 "sequencer": "00000000000044200000000000028963",
 "storageDiagnostics": {
 "batchId": "b68529f3-68cd-4744-baa4-3c0498ec19f0"
 }
 },
 "dataVersion": "",
 "metadataVersion": "1"
}
```

The following JSON shows the url of a blob that appears in the event response when a blob is created by using the Data Lake storage endpoint.

```
{
 "topic": "/subscriptions/{subscription-
id}/resourceGroups/Storage/providers/Microsoft.Storage/storageAccounts/my-storage-account",
 "subject": "/blobServices/default/containers/my-file-system/blobs/new-file.txt",
 "eventType": "Microsoft.Storage.BlobCreated",
 "eventTime": "2017-06-26T18:41:00.9584103Z",
 "id": "831e1650-001e-001b-66ab-eeb76e069631",
 "data": {
 "api": "CreateFile",
 "clientRequestId": "6d79dbfb-0e37-4fc4-981f-442c9ca65760",
 "requestId": "831e1650-001e-001b-66ab-eeb76e000000",
 "eTag": "\\"0x8D4BCC2E4835CD0\\\"",
 "contentType": "text/plain",
 "contentLength": 0,
 "contentOffset": 0,
 "blobType": "BlockBlob",
 "url": "https://my-storage-account.dfs.core.windows.net/my-file-system/new-file.txt",
 "sequencer": "00000000000044200000000000028963",
 "storageDiagnostics": {
 "batchId": "b68529f3-68cd-4744-baa4-3c0498ec19f0"
 }
 },
 "dataVersion": "2",
 "metadataVersion": "1"
}
```

If your applications use the Event Grid, you might have to modify those applications to take this into account.

## Storage Explorer

The following buttons don't yet appear in the Ribbon of Azure Storage Explorer.

BUTTON	REASON
Copy URL	Not yet implemented
Manage snapshots	Not yet implemented
Undelete	Depends on Blob storage features not yet supported with Data Lake Storage Gen2

The following buttons behave differently in your new account.

BUTTON	BLOB STORAGE BEHAVIOR	DATA LAKE STORAGE GEN2 BEHAVIOR
Folder	Folder is virtual and disappears if you don't add files to it.	Folder exists even with no files added to it.
Rename	Results in a copy and then a delete of the source blob	Renames the same blob. Far more efficient.

## Documentation

You can find guidance for using Data Lake Storage Gen2 capabilities here: [Introduction to Azure Data Lake Storage Gen2](#).

Nothing has changed with respect to where you find the guidance for all of the existing Blob storage features. That guidance is here: [Introduction to Azure Blob storage](#).

As you move between content sets, you'll notice some slight terminology differences. For example, content featured in the Data Lake Storage Gen2 content might use the term *file* and *file system* instead of *blob* and *container*. The terms *file* and *file system* are deeply rooted in the world of big data analytics where Data Lake storage has had a long history. The content contains these terms to keep it relatable to these audiences. These terms don't describe separate *things*.

## Next steps

When you are ready to upgrade your storage account to include Data Lake Storage Gen2 capabilities, see this step-by-step guide.

[Upgrade Azure Blob Storage with Azure Data Lake Storage Gen2 capabilities](#)

# Create a storage account

8/22/2022 • 22 minutes to read • [Edit Online](#)

An Azure storage account contains all of your Azure Storage data objects: blobs, files, queues, and tables. The storage account provides a unique namespace for your Azure Storage data that is accessible from anywhere in the world over HTTP or HTTPS. For more information about Azure storage accounts, see [Storage account overview](#).

In this how-to article, you learn to create a storage account using the [Azure portal](#), [Azure PowerShell](#), [Azure CLI](#), or an [Azure Resource Manager template](#).

## Prerequisites

If you don't have an Azure subscription, create a [free account](#) before you begin.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Bicep](#)
- [Template](#)

None.

Next, sign in to Azure.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Bicep](#)
- [Template](#)

Sign in to the [Azure portal](#).

## Create a storage account

A storage account is an Azure Resource Manager resource. Resource Manager is the deployment and management service for Azure. For more information, see [Azure Resource Manager overview](#).

Every Resource Manager resource, including an Azure storage account, must belong to an Azure resource group. A resource group is a logical container for grouping your Azure services. When you create a storage account, you have the option to either create a new resource group, or use an existing resource group. This how-to shows how to create a new resource group.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Bicep](#)
- [Template](#)

To create an Azure storage account with the Azure portal, follow these steps:

- From the left portal menu, select **Storage accounts** to display a list of your storage accounts. If the portal menu isn't visible, click the menu button to toggle it on.

The screenshot shows the Microsoft Azure portal interface. On the left, there's a vertical sidebar with various service links like Home, All services, and Storage accounts. The 'Storage accounts' link is highlighted with a red box. The main content area shows a grid of icons for different Azure services: Create a resource, Storage accounts, All resources, SQL virtual machines, SQL servers, SQL managed instances, Azure SQL, SQL databases, Disks (classic), and All services. Below this, there are sections for Navigate (Subscriptions, Resource groups, All resources, Dashboard), Tools (Microsoft Learn, Azure Monitor, Microsoft Defender for Cloud, Cost Management), and Useful links (Technical Documentation, Azure Services, Recent Azure Updates, Quickstart Center). At the bottom, there's an 'Azure mobile app' section with download links for the App Store and Google Play.

- On the **Storage accounts** page, select **Create**.

The screenshot shows the 'Storage accounts' list page. At the top, there's a header with a '+ Create' button, which is highlighted with a red box. Below the header is a search bar and filter options for Subscription, Resource group, and Location. The main area is a table listing four storage accounts: 'groovystorageaccount', 'hepcatstorageaccount', and 'righteousstorageacct'. Each row includes a checkbox, the account name, type, kind, resource group, location, subscription, and a three-dot ellipsis menu. At the bottom of the table, there are navigation buttons for 'Previous', 'Page 1 of 1', 'Next >', and 'Showing 1 to 4 of 4 records.' There's also a 'Give feedback' link.

Options for your new storage account are organized into tabs in the **Create a storage account** page. The following sections describe each of the tabs and their options.

### Basics tab

On the **Basics** tab, provide the essential information for your storage account. After you complete the **Basics** tab, you can choose to further customize your new storage account by setting options on the other tabs, or you can select **Review + create** to accept the default options and proceed to validate and create the account.

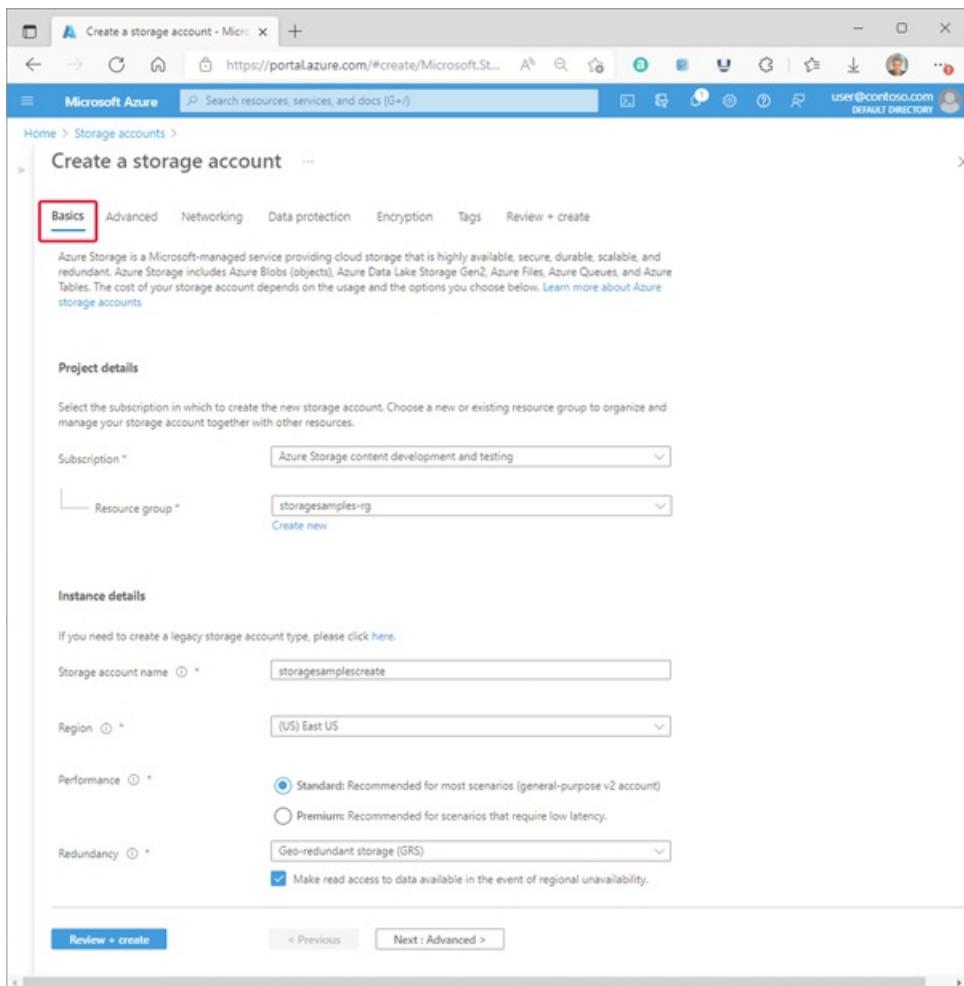
The following table describes the fields on the **Basics** tab.

SECTION	FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Project details	Subscription	Required	Select the subscription for the new storage account.

SECTION	FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Project details	Resource group	Required	Create a new resource group for this storage account, or select an existing one. For more information, see <a href="#">Resource groups</a> .
Instance details	Storage account name	Required	Choose a unique name for your storage account. Storage account names must be between 3 and 24 characters in length and may contain numbers and lowercase letters only.
Instance details	Region	Required	<p>Select the appropriate region for your storage account. For more information, see <a href="#">Regions and Availability Zones in Azure</a>.</p> <p>Not all regions are supported for all types of storage accounts or redundancy configurations. For more information, see <a href="#">Azure Storage redundancy</a>.</p> <p>The choice of region can have a billing impact. For more information, see <a href="#">Storage account billing</a>.</p>

SECTION	FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Instance details	Performance	Required	<p>Select <b>Standard</b> performance for general-purpose v2 storage accounts (default). This type of account is recommended by Microsoft for most scenarios. For more information, see <a href="#">Types of storage accounts</a>.</p> <p>Select <b>Premium</b> for scenarios requiring low latency. After selecting <b>Premium</b>, select the type of premium storage account to create. The following types of premium storage accounts are available:</p> <ul style="list-style-type: none"> <li>• <a href="#">Block blobs</a></li> <li>• <a href="#">File shares</a></li> <li>• <a href="#">Page blobs</a></li> </ul> <p>Microsoft recommends creating a general-purpose v2, premium block blob, or premium file share account for most scenarios. To select a legacy account type, use the link provided beneath <b>Instance details</b>. For more information about legacy account types, see <a href="#">Legacy storage account types</a>.</p>
Instance details	Redundancy	Required	<p>Select your desired redundancy configuration. Not all redundancy options are available for all types of storage accounts in all regions. For more information about redundancy configurations, see <a href="#">Azure Storage redundancy</a>.</p> <p>If you select a geo-redundant configuration (GRS or GZRS), your data is replicated to a data center in a different region. For read access to data in the secondary region, select <b>Make read access to data available in the event of regional unavailability</b>.</p>

The following image shows a standard configuration of the basic properties for a new storage account.



## Advanced tab

On the **Advanced** tab, you can configure additional options and modify default settings for your new storage account. Some of these options can also be configured after the storage account is created, while others must be configured at the time of creation.

The following table describes the fields on the **Advanced** tab.

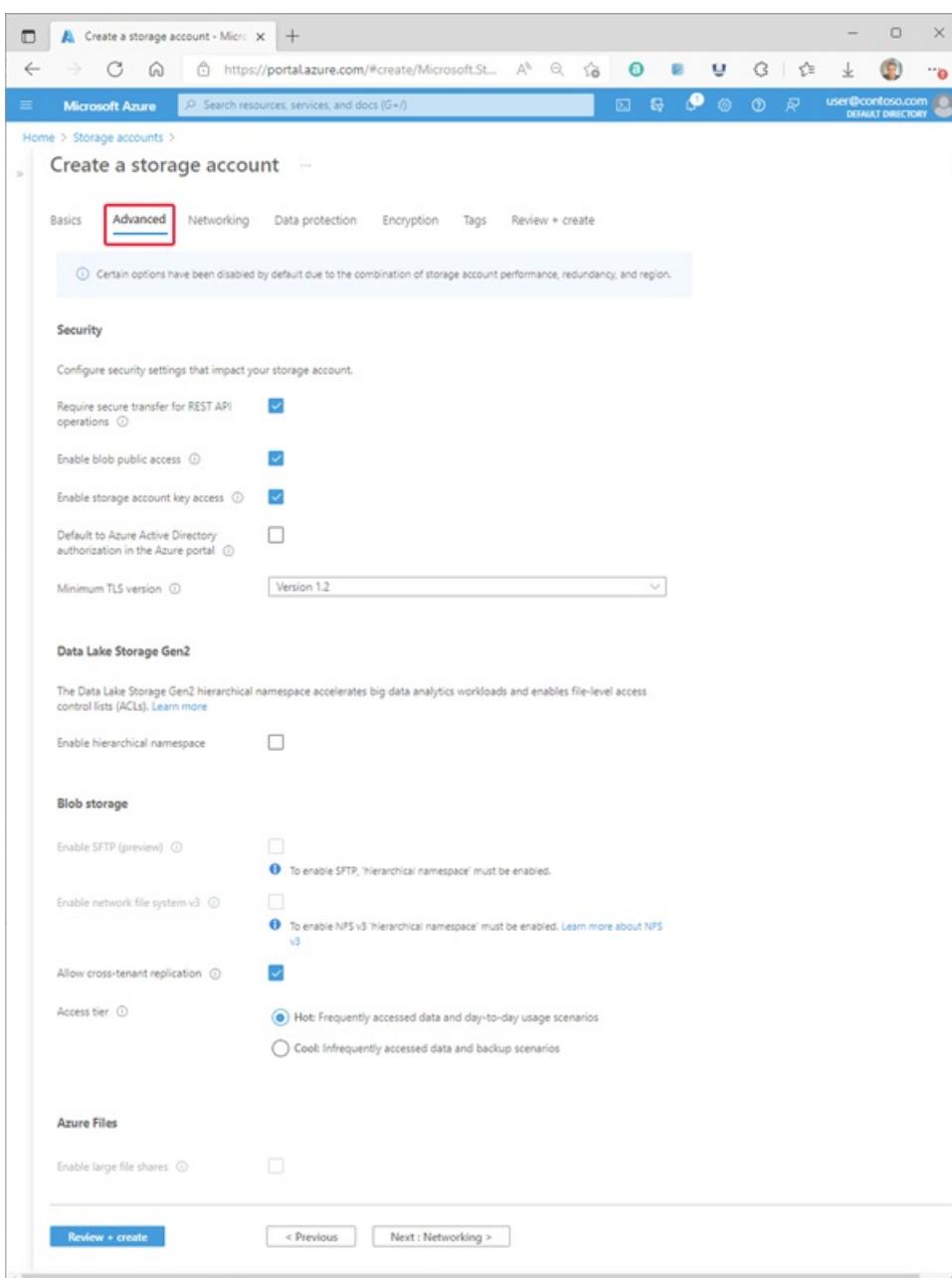
SECTION	FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Security	Require secure transfer for REST API operations	Optional	Require secure transfer to ensure that incoming requests to this storage account are made only via HTTPS (default). Recommended for optimal security. For more information, see <a href="#">Require secure transfer to ensure secure connections</a> .

SECTION	FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Security	Enable blob public access	Optional	<p>When enabled, this setting allows a user with the appropriate permissions to enable anonymous public access to a container in the storage account (default). Disabling this setting prevents all anonymous public access to the storage account. For more information, see <a href="#">Prevent anonymous public read access to containers and blobs</a>.</p> <p>Enabling blob public access does not make blob data available for public access unless the user takes the additional step to explicitly configure the container's public access setting.</p>
Security	Enable storage account key access	Optional	<p>When enabled, this setting allows clients to authorize requests to the storage account using either the account access keys or an Azure Active Directory (Azure AD) account (default). Disabling this setting prevents authorization with the account access keys. For more information, see <a href="#">Prevent Shared Key authorization for an Azure Storage account</a>.</p>
Security	Default to Azure Active Directory authorization in the Azure portal	Optional	<p>When enabled, the Azure portal authorizes data operations with the user's Azure AD credentials by default. If the user does not have the appropriate permissions assigned via Azure role-based access control (Azure RBAC) to perform data operations, then the portal will use the account access keys for data access instead. The user can also choose to switch to using the account access keys. For more information, see <a href="#">Default to Azure AD authorization in the Azure portal</a>.</p>

SECTION	FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Security	Minimum TLS version	Required	Select the minimum version of Transport Layer Security (TLS) for incoming requests to the storage account. The default value is TLS version 1.2. When set to the default value, incoming requests made using TLS 1.0 or TLS 1.1 are rejected. For more information, see <a href="#">Enforce a minimum required version of Transport Layer Security (TLS) for requests to a storage account</a> .
Data Lake Storage Gen2	Enable hierarchical namespace	Optional	To use this storage account for Azure Data Lake Storage Gen2 workloads, configure a hierarchical namespace. For more information, see <a href="#">Introduction to Azure Data Lake Storage Gen2</a> .
Blob storage	Enable SFTP	Optional	Enable the use of Secure File Transfer Protocol (SFTP) to securely transfer of data over the internet. For more information, see <a href="#">Secure File Transfer (SFTP) protocol support in Azure Blob Storage</a> .
Blob storage	Enable network file share (NFS) v3	Optional	NFS v3 provides Linux file system compatibility at object storage scale enables Linux clients to mount a container in Blob storage from an Azure Virtual Machine (VM) or a computer on-premises. For more information, see <a href="#">Network File System (NFS) 3.0 protocol support in Azure Blob storage</a> .
Blob storage	Allow cross-tenant replication	Required	By default, users with appropriate permissions can configure object replication across Azure AD tenants. To prevent replication across tenants, deselect this option. For more information, see <a href="#">Prevent replication across Azure AD tenants</a> .

Section	Field	Required or Optional	Description
Blob storage	Access tier	Required	Blob access tiers enable you to store blob data in the most cost-effective manner, based on usage. Select the hot tier (default) for frequently accessed data. Select the cool tier for infrequently accessed data. For more information, see <a href="#">Hot, Cool, and Archive access tiers for blob data</a> .
Azure Files	Enable large file shares	Optional	Available only for standard file shares with the LRS or ZRS redundancies.

The following image shows a standard configuration of the advanced properties for a new storage account.



## Networking tab

On the **Networking** tab, you can configure network connectivity and routing preference settings for your new

storage account. These options can also be configured after the storage account is created.

The following table describes the fields on the **Networking** tab.

SECTION	FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Network connectivity	Connectivity method	Required	By default, incoming network traffic is routed to the public endpoint for your storage account. You can specify that traffic must be routed to the public endpoint through an Azure virtual network. You can also configure private endpoints for your storage account. For more information, see <a href="#">Use private endpoints for Azure Storage</a> .
Network connectivity	Endpoint type	Required	Azure Storage supports two types of endpoints: standard endpoints (the default) and Azure DNS zone endpoints (preview). Within a given subscription, you can create up to 250 accounts with standard endpoints per region, and up to 5000 accounts with Azure DNS zone endpoints per region. To learn how to view the service endpoints for an existing storage account, see <a href="#">Get service endpoints for the storage account</a> .
Network routing	Routing preference	Required	The network routing preference specifies how network traffic is routed to the public endpoint of your storage account from clients over the internet. By default, a new storage account uses Microsoft network routing. You can also choose to route network traffic through the POP closest to the storage account, which may lower networking costs. For more information, see <a href="#">Network routing preference for Azure Storage</a> .

The following image shows a standard configuration of the networking properties for a new storage account.

## IMPORTANT

Azure DNS zone endpoints are currently in PREVIEW. See the [Supplemental Terms of Use for Microsoft Azure Previews](#) for legal terms that apply to Azure features that are in beta, preview, or otherwise not yet released into general availability.

## Data protection tab

On the **Data protection** tab, you can configure data protection options for blob data in your new storage account. These options can also be configured after the storage account is created. For an overview of data protection options in Azure Storage, see [Data protection overview](#).

The following table describes the fields on the **Data protection** tab.

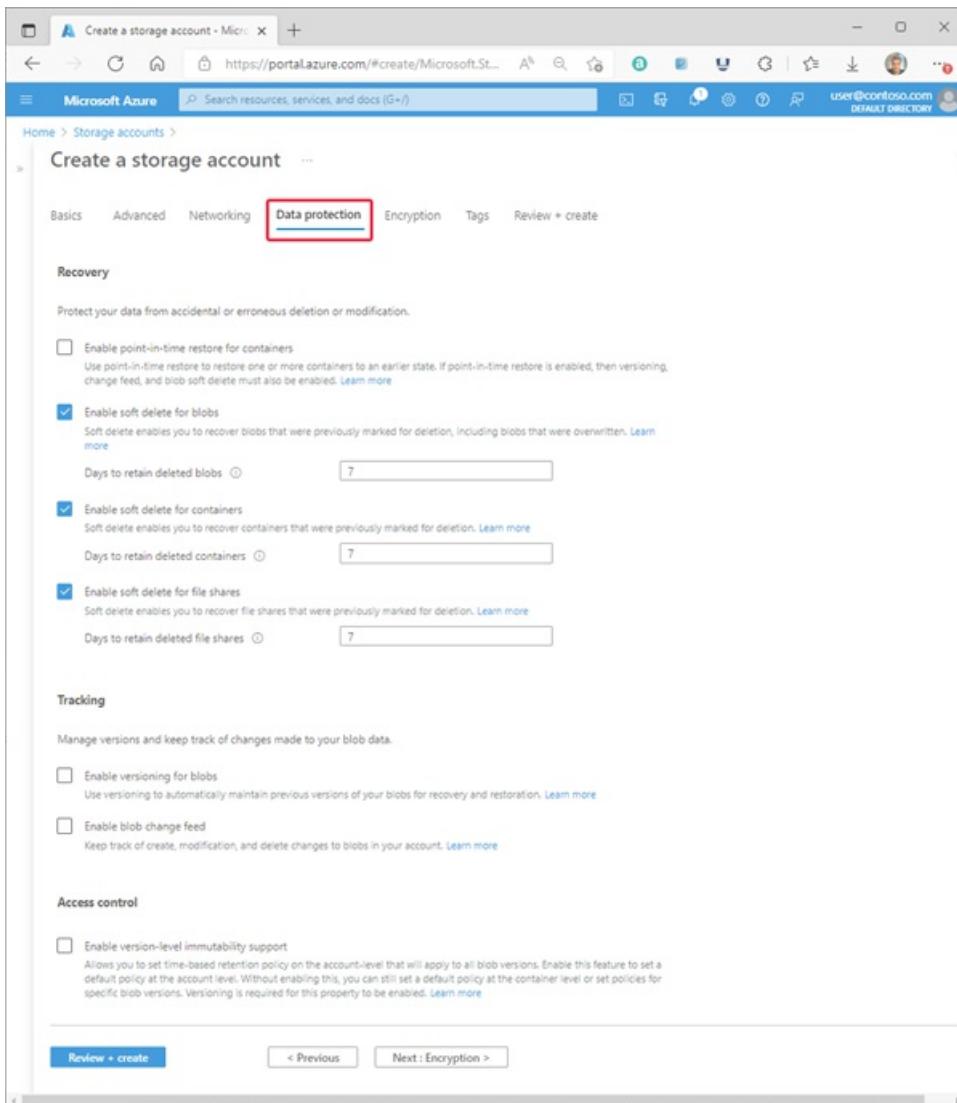
SECTION	FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Recovery	Enable point-in-time restore for containers	Optional	<p>Point-in-time restore provides protection against accidental deletion or corruption by enabling you to restore block blob data to an earlier state. For more information, see <a href="#">Point-in-time restore for block blobs</a>.</p> <p>Enabling point-in-time restore also enables blob versioning, blob soft delete, and blob change feed. These prerequisite features may have a cost impact. For more information, see <a href="#">Pricing and billing</a> for point-in-time restore.</p>

SECTION	FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Recovery	Enable soft delete for blobs	Optional	<p>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 retention period. During the retention period, you can restore a soft-deleted object to its state at the time it was deleted. For more information, see <a href="#">Soft delete for blobs</a>.</p> <p>Microsoft recommends enabling blob soft delete for your storage accounts and setting a minimum retention period of seven days.</p>
Recovery	Enable soft delete for containers	Optional	<p>Container soft delete protects a container and its contents from accidental deletes by maintaining the deleted data in the system for a specified retention period. During the retention period, you can restore a soft-deleted container to its state at the time it was deleted. For more information, see <a href="#">Soft delete for containers (preview)</a>.</p> <p>Microsoft recommends enabling container soft delete for your storage accounts and setting a minimum retention period of seven days.</p>

SECTION	FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Recovery	Enable soft delete for file shares	Optional	<p>Soft delete for file shares protects a file share and its contents from accidental deletes by maintaining the deleted data in the system for a specified retention period. During the retention period, you can restore a soft-deleted file share to its state at the time it was deleted. For more information, see <a href="#">Prevent accidental deletion of Azure file shares</a>.</p> <p>Microsoft recommends enabling soft delete for file shares for Azure Files workloads and setting a minimum retention period of seven days.</p>
Tracking	Enable versioning for blobs	Optional	<p>Blob versioning automatically saves the state of a blob in a previous version when the blob is overwritten. For more information, see <a href="#">Blob versioning</a>.</p> <p>Microsoft recommends enabling blob versioning for optimal data protection for the storage account.</p>
Tracking	Enable blob change feed	Optional	<p>The blob change feed provides transaction logs of all changes to all blobs in your storage account, as well as to their metadata. For more information, see <a href="#">Change feed support in Azure Blob Storage</a>.</p>

Section	Field	Required or Optional	Description
Access control	Enable version-level immutability support	Optional	Enable support for immutability policies that are scoped to the blob version. If this option is selected, then after you create the storage account, you can configure a default time-based retention policy for the account or for the container, which blob versions within the account or container will inherit by default. For more information, see <a href="#">Enable version-level immutability support on a storage account</a> .

The following image shows a standard configuration of the data protection properties for a new storage account.

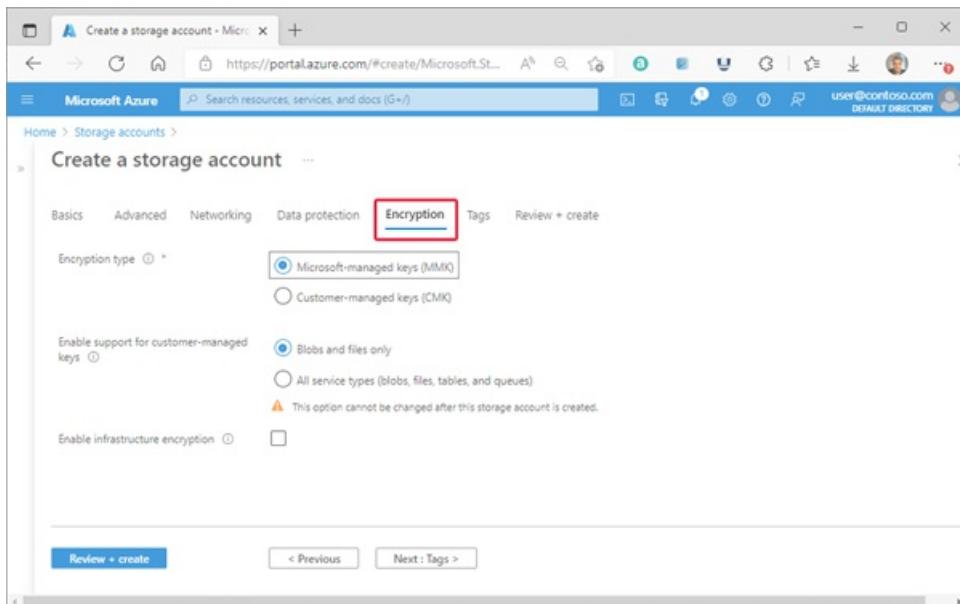


## Encryption tab

On the **Encryption** tab, you can configure options that relate to how your data is encrypted when it is persisted to the cloud. Some of these options can be configured only when you create the storage account.

FIELD	REQUIRED OR OPTIONAL	DESCRIPTION
Encryption type	Required	By default, data in the storage account is encrypted by using Microsoft-managed keys. You can rely on Microsoft-managed keys for the encryption of your data, or you can manage encryption with your own keys. For more information, see <a href="#">Azure Storage encryption for data at rest</a> .
Enable support for customer-managed keys	Required	By default, customer managed keys can be used to encrypt only blobs and files. Set this option to <b>All service types (blobs, files, tables, and queues)</b> to enable support for customer-managed keys for all services. You are not required to use customer-managed keys if you choose this option. For more information, see <a href="#">Customer-managed keys for Azure Storage encryption</a> .
Encryption key	Required if <b>Encryption type</b> field is set to <b>Customer-managed keys</b> .	If you choose <b>Select a key vault and key</b> , you are presented with the option to navigate to the key vault and key that you wish to use. If you choose <b>Enter key from URI</b> , then you are presented with a field to enter the key URI and the subscription.
User-assigned identity	Required if <b>Encryption type</b> field is set to <b>Customer-managed keys</b> .	If you are configuring customer-managed keys at create time for the storage account, you must provide a user-assigned identity to use for authorizing access to the key vault.
Enable infrastructure encryption	Optional	By default, infrastructure encryption is not enabled. Enable infrastructure encryption to encrypt your data at both the service level and the infrastructure level. For more information, see <a href="#">Create a storage account with infrastructure encryption enabled for double encryption of data</a> .

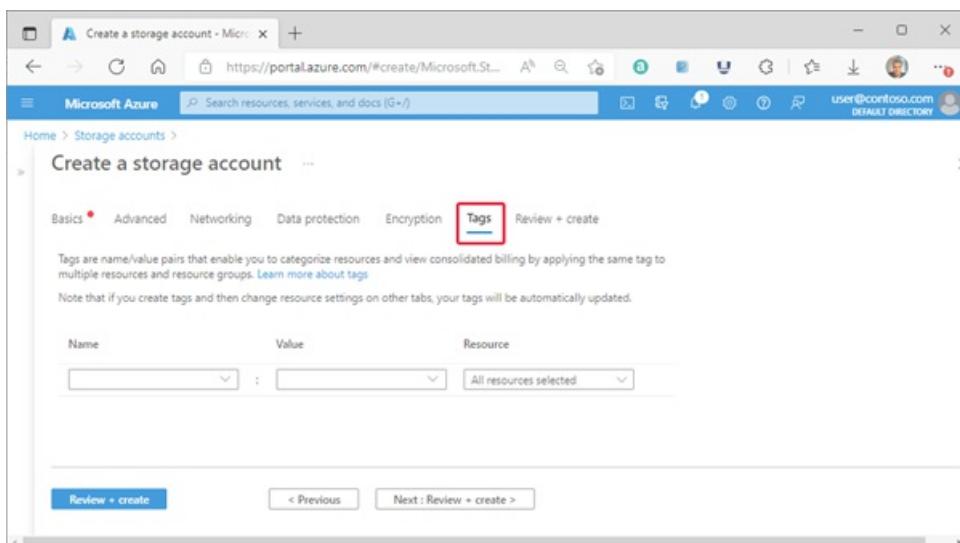
The following image shows a standard configuration of the encryption properties for a new storage account.



## Tags tab

On the **Tags** tab, you can specify Resource Manager tags to help organize your Azure resources. For more information, see [Tag resources, resource groups, and subscriptions for logical organization](#).

The following image shows a standard configuration of the index tag properties for a new storage account.



## Review + create tab

When you navigate to the **Review + create** tab, Azure runs validation on the storage account settings that you have chosen. If validation passes, you can proceed to create the storage account.

If validation fails, then the portal indicates which settings need to be modified.

The following image shows the **Review** tab data prior to the creation of a new storage account.

The screenshot shows the Microsoft Azure portal interface for creating a new storage account. The page title is 'Create a storage account'. At the top, there's a green bar with the message 'Validation passed'. Below it, the 'Review + create' button is highlighted with a red box. The configuration details are organized into sections: Basics, Advanced, Networking, Data protection, and Encryption. The 'Review + create' button is located at the bottom of the configuration area.

## Delete a storage account

Deleting a storage account deletes the entire account, including all data in the account. Be sure to back up any data you want to save before you delete the account.

Under certain circumstances, a deleted storage account may be recovered, but recovery is not guaranteed. For more information, see [Recover a deleted storage account](#).

If you try to delete a storage account associated with an Azure virtual machine, you may get an error about the storage account still being in use. For help troubleshooting this error, see [Troubleshoot errors when you delete storage accounts](#).

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Bicep](#)
- [Template](#)

1. Navigate to the storage account in the [Azure portal](#).
2. Click **Delete**.

Alternately, you can delete the resource group, which deletes the storage account and any other resources in that resource group. For more information about deleting a resource group, see [Delete resource group and resources](#).

## Next steps

- [Storage account overview](#)
- [Upgrade to a general-purpose v2 storage account](#)
- [Move a storage account to another region](#)
- [Recover a deleted storage account](#)

# Upgrade to a general-purpose v2 storage account

8/22/2022 • 11 minutes to read • [Edit Online](#)

General-purpose v2 storage accounts support the latest Azure Storage features and incorporate all of the functionality of general-purpose v1 and Blob storage accounts. General-purpose v2 accounts are recommended for most storage scenarios. General-purpose v2 accounts deliver the lowest per-gigabyte capacity prices for Azure Storage, as well as industry-competitive transaction prices. General-purpose v2 accounts support default account access tiers of hot or cool and blob level tiering between hot, cool, or archive.

Upgrading to a general-purpose v2 storage account from your general-purpose v1 or Blob storage accounts is straightforward. You can upgrade using the Azure portal, PowerShell, or Azure CLI. There is no downtime or risk of data loss associated with upgrading to a general-purpose v2 storage account. The account upgrade happens via a simple Azure Resource Manager operation that changes the account type.

## IMPORTANT

Upgrading a general-purpose v1 or Blob storage account to general-purpose v2 is permanent and cannot be undone.

## NOTE

Although Microsoft recommends general-purpose v2 accounts for most scenarios, Microsoft will continue to support general-purpose v1 accounts for new and existing customers. You can create general-purpose v1 storage accounts in new regions whenever Azure Storage is available in those regions. Microsoft does not currently have a plan to deprecate support for general-purpose v1 accounts and will provide at least one year's advance notice before deprecating any Azure Storage feature. Microsoft will continue to provide security updates for general-purpose v1 accounts, but no new feature development is expected for this account type.

For new Azure regions that have come online after October 1, 2020, pricing for general-purpose v1 accounts has changed and is equivalent to pricing for general-purpose v2 accounts in those regions. Pricing for general-purpose v1 accounts in Azure regions that existed prior to October 1, 2020 has not changed. For pricing details for general-purpose v1 accounts in a specific region, see the Azure Storage pricing page. Choose your region, and then next to **Pricing offers**, select **Other**.

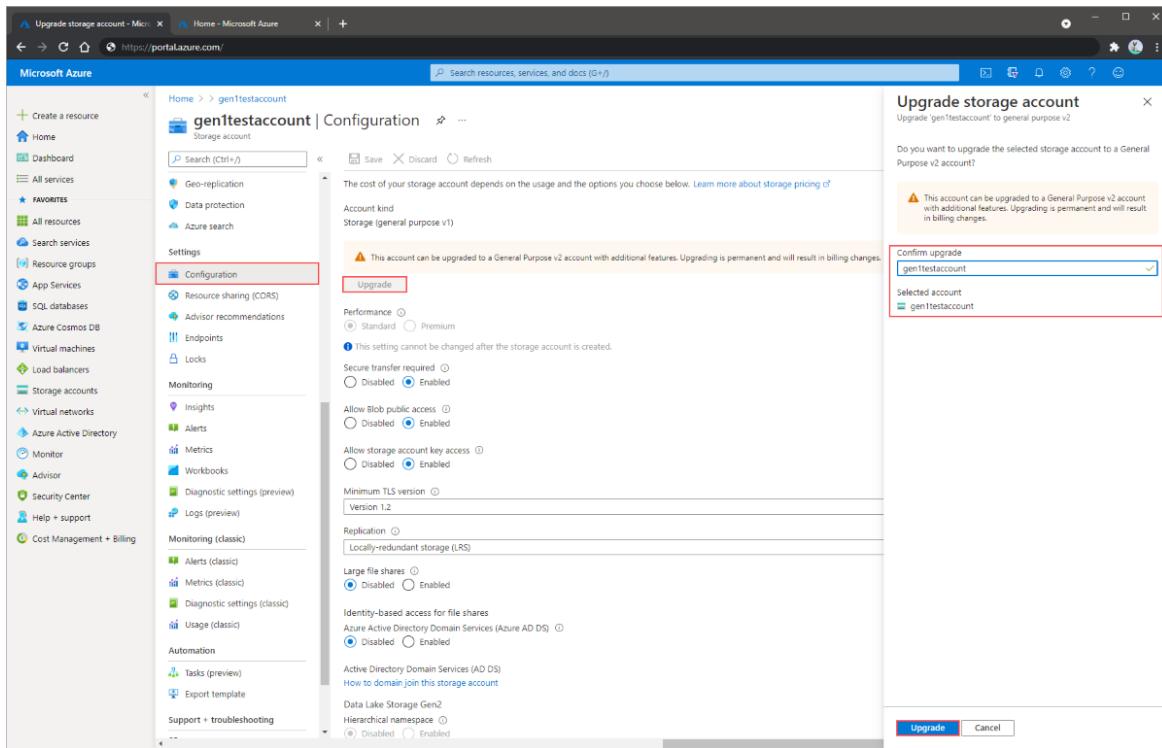
## Upgrade an account

To upgrade a general-purpose v1 or Blob storage account to a general-purpose v2 account, use Azure portal, PowerShell, or Azure CLI.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

1. Sign in to the [Azure portal](#).
2. Navigate to your storage account.
3. In the **Settings** section, select **Configuration**.
4. Under **Account kind**, select on **Upgrade**.
5. Under **Confirm upgrade**, enter the name of your account.

## 6. Select Upgrade at the bottom of the blade.



## Specify an access tier for blob data

General-purpose v2 accounts support all Azure storage services and data objects, but access tiers are available only to block blobs within Blob storage. When you upgrade to a general-purpose v2 storage account, you can specify a default account access tier of hot or cool, which indicates the default tier your blob data will be uploaded as if the individual blob access tier parameter is not specified.

Blob access tiers enable you to choose the most cost-effective storage based on your anticipated usage patterns. Block blobs can be stored in a hot, cool, or archive tiers. For more information on access tiers, see [Azure Blob storage: Hot, Cool, and Archive storage tiers](#).

By default, a new storage account is created in the hot access tier, and a general-purpose v1 storage account can be upgraded to either the hot or cool account tier. If an account access tier is not specified on upgrade, it will be upgraded to hot by default. If you are exploring which access tier to use for your upgrade, consider your current data usage scenario. There are two typical user scenarios for migrating to a general-purpose v2 account:

- You have an existing general-purpose v1 storage account and want to evaluate an upgrade to a general-purpose v2 storage account, with the right storage access tier for blob data.
- You have decided to use a general-purpose v2 storage account or already have one and want to evaluate whether you should use the hot or cool storage access tier for blob data.

In both cases, the first priority is to estimate the cost of storing, accessing, and operating on your data stored in a general-purpose v2 storage account and compare that against your current costs.

## Pricing and billing

Upgrading a v1 storage account to a general-purpose v2 account is free. You may specify the desired account tier during the upgrade process. If an account tier is not specified on upgrade, the default account tier of the upgraded account will be **Hot**. However, changing the storage access tier after the upgrade may result in changes to your bill so it is recommended to specify the new account tier during upgrade.

All storage accounts use a pricing model for blob storage based on the tier of each blob. When using a storage account, the following billing considerations apply:

- **Storage costs:** In addition to the amount of data stored, the cost of storing data varies depending on the storage 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 storage access tier, you are charged a per-gigabyte data access charge for reads.
- **Transaction costs:** There is 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 storage access tier:** Changing the account storage access tier from cool to hot incurs a charge equal to reading all the data existing in the storage account. However, changing the account access tier from hot to cool incurs a charge equal to writing all the data into the cool tier (GPv2 accounts only).

**NOTE**

For more information on the pricing model for storage accounts, see [Azure Storage Pricing](#) page. For more information on outbound data transfer charges, see [Data Transfers Pricing Details](#) page.

### Estimate costs for your current usage patterns

To estimate the cost of storing and accessing blob data in a general-purpose v2 storage account in a particular tier, evaluate your existing usage pattern or approximate your expected usage pattern. In general, you want to know:

- Your Blob storage consumption, in gigabytes, including:
  - How much data is being stored in the storage account?
  - How does the data volume change on a monthly basis; does new data constantly replace old data?
- The primary access pattern for your Blob storage data, including:
  - How much data is being read from and written to the storage account?
  - How many read operations versus write operations occur on the data in the storage account?

To decide on the best access tier for your needs, it can be helpful to determine your blob data capacity, and how that data is being used. This can be best done by looking at the monitoring metrics for your account.

### Monitoring existing storage accounts

To monitor your existing storage accounts and gather this data, you can make use of Azure Storage Analytics, which performs logging and provides metrics data for a storage account. Storage Analytics can store metrics that include aggregated transaction statistics and capacity data about requests to the storage service for GPv1, GPv2, and Blob storage account types. This data is stored in well-known tables in the same storage account.

For more information, see [About Storage Analytics Metrics](#) and [Storage Analytics Metrics Table Schema](#)

**NOTE**

Blob storage accounts expose the Table service endpoint only for storing and accessing the metrics data for that account.

To monitor the storage consumption for Blob storage, you need to enable the capacity metrics. With this enabled, capacity data is recorded daily for a storage account's Blob service and recorded as a table entry that is written to the `$MetricsCapacityBlob` table within the same storage account.

To monitor data access patterns for Blob storage, you need to enable the hourly transaction metrics from the API. With hourly transaction metrics enabled, per API transactions are aggregated every hour, and recorded as a table entry that is written to the `$MetricsHourPrimaryTransactionsBlob` table within the same storage account. The `$MetricsHourSecondaryTransactionsBlob` table records the transactions to the secondary endpoint when using RA-GRS storage accounts.

#### NOTE

If you have a general-purpose storage account in which you have stored page blobs and virtual machine disks, or queues, files, or tables, alongside block and append blob data, this estimation process is not applicable. The capacity data does not differentiate block blobs from other types, and does not give capacity data for other data types. If you use these types, an alternative methodology is to look at the quantities on your most recent bill.

To get a good approximation of your data consumption and access pattern, we recommend you choose a retention period for the metrics that is representative of your regular usage and extrapolate. One option is to retain the metrics data for seven days and collect the data every week, for analysis at the end of the month. Another option is to retain the metrics data for the last 30 days and collect and analyze the data at the end of the 30-day period.

For details on enabling, collecting, and viewing metrics data, see [Storage analytics metrics](#).

#### NOTE

Storing, accessing, and downloading analytics data is also charged just like regular user data.

## Utilizing usage metrics to estimate costs

### Capacity costs

The latest entry in the capacity metrics table `$MetricsCapacityBlob` with the row key '`data`' shows the storage capacity consumed by user data. The latest entry in the capacity metrics table `$MetricsCapacityBlob` with the row key '`analytics`' shows the storage capacity consumed by the analytics logs.

This total capacity consumed by both user data and analytics logs (if enabled) can then be used to estimate the cost of storing data in the storage account. The same method can also be used for estimating storage costs in GPv1 storage accounts.

### Transaction costs

The sum of '`TotalBillableRequests`', across all entries for an API in the transaction metrics table indicates the total number of transactions for that particular API. *For example*, the total number of '`GetBlob`' transactions in a given period can be calculated by the sum of total billable requests for all entries with the row key '`user;GetBlob`'.

In order to estimate transaction costs for Blob storage accounts, you need to break down the transactions into three groups since they are priced differently.

- Write transactions such as '`PutBlob`', '`PutBlock`', '`PutBlockList`', '`AppendBlock`', '`ListBlobs`', '`ListContainers`', '`CreateContainer`', '`SnapshotBlob`', and '`CopyBlob`'.
- Delete transactions such as '`DeleteBlob`' and '`DeleteContainer`'.
- All other transactions.

In order to estimate transaction costs for GPv1 storage accounts, you need to aggregate all transactions irrespective of the operation/API.

### Data access and geo-replication data transfer costs

While storage analytics does not provide the amount of data read from and written to a storage account, it can be roughly estimated by looking at the transaction metrics table. The sum of '`TotalIngress`' across all entries for an API in the transaction metrics table indicates the total amount of ingress data in bytes for that particular API.

Similarly the sum of '*TotalEgress*' indicates the total amount of egress data, in bytes.

In order to estimate the data access costs for Blob storage accounts, you need to break down the transactions into two groups.

- The amount of data retrieved from the storage account can be estimated by looking at the sum of '*TotalEgress*' for primarily the '*GetBlob*' and '*CopyBlob*' operations.
- The amount of data written to the storage account can be estimated by looking at the sum of '*TotalIngress*' for primarily the '*PutBlob*', '*PutBlock*', '*CopyBlob*' and '*AppendBlock*' operations.

The cost of geo-replication data transfer for Blob storage accounts can also be calculated by using the estimate for the amount of data written when using a GRS or RA-GRS storage account.

#### NOTE

For a more detailed example about calculating the costs for using the hot or cool storage access tier, take a look at the FAQ titled '*What are Hot and Cool access tiers and how should I determine which one to use?*' in the [Azure Storage Pricing Page](#).

## Next steps

- [Storage account overview](#)
- [Create a storage account](#)
- [Move an Azure Storage account to another region](#)
- [Recover a deleted storage account](#)

# Recover a deleted storage account

8/22/2022 • 2 minutes to read • [Edit Online](#)

A deleted storage account may be recovered in some cases from within the Azure portal. To recover a storage account, the following conditions must be true:

- The storage account was deleted within the past 14 days.
- The storage account was created with the Azure Resource Manager deployment model.
- A new storage account with the same name has not been created since the original account was deleted.
- The user who is recovering the storage account must be assigned an Azure RBAC role that provides the `Microsoft.Storage/storageAccounts/write` permission. For information about built-in Azure RBAC roles that provide this permission, see [Azure built-in roles](#).

Before you attempt to recover a deleted storage account, make sure that the resource group for that account exists. If the resource group was deleted, you must recreate it. Recovering a resource group is not possible. For more information, see [Manage resource groups](#).

If the deleted storage account used customer-managed keys with Azure Key Vault and the key vault has also been deleted, then you must restore the key vault before you restore the storage account. For more information, see [Azure Key Vault recovery overview](#).

## IMPORTANT

Recovery of a deleted storage account is not guaranteed. Recovery is a best-effort attempt. Microsoft recommends locking resources to prevent accidental account deletion. For more information about resource locks, see [Lock resources to prevent changes](#).

Another best practice to avoid accidental account deletion is to limit the number of users who have permissions to delete an account via role-based access control (Azure RBAC). For more information, see [Best practices for Azure RBAC](#).

## Recover a deleted account from the Azure portal

To restore a deleted storage account from within another storage account, follow these steps:

1. Navigate to the list of your storage accounts in the Azure portal.
2. Select the **Restore** button to open the **Restore deleted account** pane.
3. Select the subscription for the account that you want to recover from the **Subscription** drop-down.
4. From the dropdown, select the account to recover, as shown in the following image. If the storage account that you want to recover is not in the dropdown, then it cannot be recovered.

## Restore deleted account

X

**i** Please note that the successful recovery of any storage account is not guaranteed, even if it is listed below. If a storage account was deleted more than 14 days ago, is a classic account, or does not appear below, then it cannot be recovered.

Select the subscription associated with the deleted storage account.

Subscription

Azure Storage content development and testing

Please confirm the resource group containing the storage account you wish to recover still exists. If it has been deleted, please recreate it before recovering your storage account.

Deleted storage account (last 14 days) \*

Select account to recover

Resource group: storagesamples-rg

Storage account: storagesamplesdeleted

Location: East US, Creation: 6/23/2022, 11:16:39 AM, Deletion: 6/23/2022, 11:19:29 AM



Restore

Close

5. Select the **Restore** button to recover the account. The portal displays a notification that the recovery is in progress.

## Recover a deleted account via a support ticket

1. In the Azure portal, navigate to **Help + support**.
2. Select **New support request**.
3. On the **Basics** tab, in the **Issue type** field, select **Technical**.
4. In the **Subscription** field, select the subscription that contained the deleted storage account.
5. In the **Service** field, select **Storage Account Management**.
6. In the **Resource** field, select any storage account resource. The deleted storage account will not appear in the list.
7. Add a brief summary of the issue.
8. In the **Problem type** field, select **Deletion and Recovery**.
9. In the **Problem subtype** field, select **Recover deleted storage account**. The following image shows an example of the **Basics** tab being filled out:

## New support request



Basics    Solutions    Details    Review + create

---

Create a new support request to get assistance with billing, subscription, technical (including advisory) or quota management issues.

Complete the Basics tab by selecting the options that best describe your problem. Providing detailed, accurate information can help to solve your issues faster.

* Issue type	Technical
* Subscription	Azure Storage content development and t... Can't find your subscription? <a href="#">Show more</a> ⓘ
* Service	<input checked="" type="radio"/> My services <input type="radio"/> All services Storage Account Management
* Resource	storagesamples
* Summary	recover deleted storage account
* Problem type	Deletion and Recovery
* Problem subtype	Recover deleted storage account

---

[Next: Solutions >>](#)



10. Next, navigate to the **Solutions** tab, and select **Customer-Controlled Storage Account Recovery**, as shown in the following image:

## New support request

X

Basics    Solutions    Details    Review + create

Want a solution right now?

Try following the recommended steps below. These solutions are written by Azure engineers, and will solve most common issues.



Recommended Solution

### Recommended Steps

Azure Storage now provides users a self-service storage account recovery capability.

#### Conditions for a storage account to be recoverable:

- A new storage object with the same name has not been re-created since deletion.
- The storage account was deleted in the last 14 days, including today. If the storage account was deleted prior to that, it can't be recovered.
- It is not a classic storage account.

#### Prerequisites:

- Ensure that the Resource Group is created first, if it has been deleted.
- Ensure that the KeyVault key is recovered first for storage accounts with CMK.

#### Disclaimer:

- There's no guarantee that recovery will always succeed. Recovery is a best effort rather than a guarantee.
- We strongly recommend using a combination of [ARM resource locks](#) and [Azure RBAC](#) permissions to prevent accidental account deletion.

### Customer-Controlled Storage Account Recovery

Show less ^

Was this helpful? Yes No

<< Previous: Basics

Next: Details >>



11. From the dropdown, select the account to recover, as shown in the following image. If the storage account that you want to recover is not in the dropdown, then it cannot be recovered.

## Recover deleted account

X

Review a list of storage accounts deleted in the past 14 days below for the following subscription:

**Azure Storage content development and testing.**

Please confirm the resource group containing the storage account you wish to recover still exists. If it has been deleted, please recreate it before recovering your storage account.

Deleted storage accounts (last 14 days) \*

Select account to recover

Resource group: storagesamples-rg

Storage account: storagesamplesrecover

Location: West US, Creation: 12/10/2020, 12:04:56 PM, Deletion: 12/10/2020, 12:43:09 PM

Recover

Close

12. Select the **Recover** button to restore the account. The portal displays a notification that the recovery is in progress.

## Next steps

- [Storage account overview](#)
- [Create a storage account](#)
- [Move an Azure Storage account to another region](#)

# Get storage account configuration information

8/22/2022 • 4 minutes to read • [Edit Online](#)

This article shows how to get configuration information and properties for an Azure Storage account by using the Azure portal, PowerShell, or Azure CLI.

## Get the resource ID for a storage account

Every Azure Resource Manager resource has an associated resource ID that uniquely identifies it. Certain operations require that you provide the resource ID. You can get the resource ID for a storage account by using the Azure portal, PowerShell, or Azure CLI.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To display the Azure Resource Manager resource ID for a storage account in the Azure portal, follow these steps:

1. Navigate to your storage account in the Azure portal.
2. On the **Overview** page, in the **Essentials** section, select the **JSON View** link.
3. The resource ID for the storage account is displayed at the top of the page.



You can also get the resource ID for a storage account by calling the [Storage Accounts - Get Properties](#) operation in the REST API.

For more information about types of resources managed by Azure Resource Manager, see [Resource providers and resource types](#).

## Get the account type, location, or replication SKU for a storage account

The account type, location, and replication SKU are some of the properties available on a storage account. You can use the Azure portal, PowerShell, or Azure CLI to view these values.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To view the account type, location, or replication SKU for a storage account in the Azure portal, follow these steps:

1. Navigate to your storage account in the Azure portal.
2. Locate these properties on the **Overview** page, in the **Essentials** section

^ Essentials	
Resource group ( <a href="#">change</a> )	: storagesamples-rg
Location	: Canada Central
Primary/Secondary Location	: Primary: Canada Central, Secondary: Canada East
Subscription ( <a href="#">change</a> )	: <subscription-name>
Subscription ID	: <subscription-id>
Disk state	: Primary: Available, Secondary: Available
Performance/Access tier : Standard/Hot	
Replication	: Read-access geo-redundant storage (RA-GRS)
Account kind	: StorageV2 (general purpose v2)
Provisioning state	: Succeeded
Created	: 9/11/2020, 9:55:16 AM

## Get service endpoints for the storage account

The service endpoints for a storage account provide the base URL for any blob, queue, table, or file object in Azure Storage. Use this base URL to construct the address for any given resource.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To get the service endpoints for a storage account in the Azure portal, follow these steps:

1. Navigate to your storage account in the Azure portal.
2. In the **Settings** section, locate the **Endpoints** setting.
3. On the **Endpoints** page, you'll see the service endpoint for each Azure Storage service, as well as the resource ID.

Service	Resource ID	Endpoint URL
Blob service	/subscriptions/<subscription-id>/resourceGroups/storagesamples-rg/providers/Microsoft.Storage/storageAccounts/storagesamplesdnszone4	https://storagesamplesdnszone4.z10.blob.storage.azure.net/
File service	/subscriptions/<subscription-id>/resourceGroups/storagesamples-rg/providers/Microsoft.Storage/storageAccounts/storagesamplesdnszone4	https://storagesamplesdnszone4.z10.file.storage.azure.net/
Queue service	/subscriptions/<subscription-id>/resourceGroups/storagesamples-rg/providers/Microsoft.Storage/storageAccounts/storagesamplesdnszone4	https://storagesamplesdnszone4.z10.queue.storage.azure.net/

If the storage account is geo-replicated, the secondary endpoints will also appear on this page.

## Get a connection string for the storage account

You can use a connection string to authorize access to Azure Storage with the account access keys (Shared Key authorization). To learn more about connection strings, see [Configure Azure Storage connection strings](#).

## Protect your access keys

Your storage account access keys are similar to a root password for your storage account. Always be careful to protect your access keys. Use Azure Key Vault to manage and rotate your keys securely. Avoid distributing access keys to other users, hard-coding them, or saving them anywhere in plain text that is accessible to others. Rotate your keys if you believe they may have been compromised.

#### **NOTE**

Microsoft recommends using Azure Active Directory (Azure AD) to authorize requests against blob and queue data if possible, rather than using the account keys (Shared Key authorization). Authorization with Azure AD provides superior security and ease of use over Shared Key authorization.

To protect an Azure Storage account with Azure AD Conditional Access policies, you must disallow Shared Key authorization for the storage account. For more information about how to disallow Shared Key authorization, see [Prevent Shared Key authorization for an Azure Storage account](#).

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To get a connection string in the Azure portal, follow these steps:

1. Navigate to your storage account in the Azure portal.
2. In the **Security + networking** section, locate the **Access keys** setting.
3. To display the account keys and associated connection strings, select the **Show keys** button at the top of the page.
4. To copy a connection string to the clipboard, select the **Copy** button to the right of the connection string.

## Next steps

- [Storage account overview](#)
- [Create a storage account](#)

# Move an Azure Storage account to another region

8/22/2022 • 5 minutes to read • [Edit Online](#)

To move a storage account, create a copy of your storage account in another region. Then, move your data to that account by using AzCopy, or another tool of your choice.

In this article, you'll learn how to:

- Export a template.
- Modify the template by adding the target region and storage account name.
- Deploy the template to create the new storage account.
- Configure the new storage account.
- Move data to the new storage account.
- Delete the resources in the source region.

## Prerequisites

- Ensure that the services and features that your account uses are supported in the target region.
- For preview features, ensure that your subscription is allowlisted for the target region.

## Prepare

To get started, export, and then modify a Resource Manager template.

### Export a template

This template contains settings that describe your storage account.

- [Portal](#)
- [PowerShell](#)

To export a template by using Azure portal:

1. Sign in to the [Azure portal](#).
2. Select **All resources** and then select your storage account.
3. Select > **Automation** > **Export template**.
4. Choose **Download** in the **Export template** blade.
5. Locate the .zip file that you downloaded from the portal, and unzip that file to a folder of your choice.

This zip file contains the json files that comprise the template and scripts to deploy the template.

### Modify the template

Modify the template by changing the storage account name and region.

- [Portal](#)
- [PowerShell](#)

To deploy the template by using Azure portal:

1. In the Azure portal, select **Create a resource**.
2. In **Search the Marketplace**, type **template deployment**, and then press **ENTER**.
3. Select **Template deployment**.

NAME	PUBLISHER	CATEGORY
Template deployment	Microsoft	Compute
Radware Alteon VA - deployment template	Radware	Compute

4. Select **Create**.
5. Select **Build your own template in the editor**.
6. Select **Load file**, and then follow the instructions to load the **template.json** file that you downloaded in the last section.
7. In the **template.json** file, name the target storage account by setting the default value of the storage account name. This example sets the default value of the storage account name to `mytargetaccount`.

```

"$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
"contentVersion": "1.0.0.0",
"parameters": {
 "storageAccounts_mysourceaccount_name": {
 "defaultValue": "mytargetaccount",
 "type": "String"
 }
},

```

8. Edit the **location** property in the **template.json** file to the target region. This example sets the target region to `centralus`.

```

"resources": [
 {
 "type": "Microsoft.Storage/storageAccounts",
 "apiVersion": "2019-04-01",
 "name": "[parameters('storageAccounts_mysourceaccount_name')]",
 "location": "centralus"
 }
]

```

To obtain region location codes, see [Azure Locations](#). The code for a region is the region name with no spaces, Central US = `centralus`.

## Move

Deploy the template to create a new storage account in the target region.

- [Portal](#)
- [PowerShell](#)

1. Save the **template.json** file.

2. Enter or select the property values:

- **Subscription:** Select an Azure subscription.
- **Resource group:** Select **Create new** and give the resource group a name.
- **Location:** Select an Azure location.

3. Click the **I agree to the terms and conditions stated above** checkbox, and then click the **Select Purchase** button.

**TIP**

If you receive an error which states that the XML specified is not syntactically valid, compare the JSON in your template with the schemas described in the [Azure Resource Manager documentation](#).

## Configure the new storage account

Some features won't export to a template, so you'll have to add them to the new storage account.

The following table lists these features along with guidance for adding them to your new storage account.

FEATURE	GUIDANCE
Lifecycle management policies	<a href="#">Manage the Azure Blob storage lifecycle</a>
Static websites	<a href="#">Host a static website in Azure Storage</a>
Event subscriptions	<a href="#">Reacting to Blob storage events</a>
Alerts	<a href="#">Create, view, and manage activity log alerts by using Azure Monitor</a>
Content Delivery Network (CDN)	<a href="#">Use Azure CDN to access blobs with custom domains over HTTPS</a>

**NOTE**

If you set up a CDN for the source storage account, just change the origin of your existing CDN to the primary blob service endpoint (or the primary static website endpoint) of your new account.

## Move data to the new storage account

AzCopy is the preferred tool to move your data over. It's optimized for performance. One way that it's faster, is that data is copied directly between storage servers, so AzCopy doesn't use the network bandwidth of your computer. Use AzCopy at the command line or as part of a custom script. See [Get started with AzCopy](#).

You can also use Azure Data Factory to move your data over. It provides an intuitive user interface. To use Azure Data Factory, see any of these links:

- [Copy data to or from Azure Blob storage by using Azure Data Factory](#)
- [Copy data to or from Azure Data Lake Storage Gen2 using Azure Data Factory](#)
- [Copy data from or to Azure Files by using Azure Data Factory](#)
- [Copy data to and from Azure Table storage by using Azure Data Factory](#)

## Discard or clean up

After the deployment, if you want to start over, you can delete the target storage account, and repeat the steps described in the [Prepare](#) and [Move](#) sections of this article.

To commit the changes and complete the move of a storage account, delete the source storage account.

- [Portal](#)
- [PowerShell](#)

To remove a storage account by using the Azure portal:

1. In the Azure portal, expand the menu on the left side to open the menu of services, and choose **Storage accounts** to display the list of your storage accounts.
2. Locate the target storage account to delete, and right-click the **More** button (...) on the right side of the listing.
3. Select **Delete**, and confirm.

## Next steps

In this tutorial, you moved an Azure storage account from one region to another and cleaned up the source resources. To learn more about moving resources between regions and disaster recovery in Azure, refer to:

- [Move resources to a new resource group or subscription](#)
- [Move Azure VMs to another region](#)

# Upgrade Azure Blob Storage with Azure Data Lake Storage Gen2 capabilities

8/22/2022 • 8 minutes to read • [Edit Online](#)

This article helps you to enable a hierarchical namespace and unlock capabilities such as file and directory-level security and faster operations. These capabilities are widely used by big data analytics workloads and are referred to collectively as Azure Data Lake Storage Gen2.

To learn more about these capabilities and evaluate the impact of this upgrade on workloads, applications, costs, service integrations, tools, features, and documentation, see [Upgrading Azure Blob Storage with Azure Data Lake Storage Gen2 capabilities](#).

## IMPORTANT

An upgrade is one-way. There's no way to revert your account once you've performed the upgrade. We recommend that you validate your upgrade in a nonproduction environment.

## Review feature support

Your account might be configured to use features that aren't yet supported in Data Lake Storage Gen2 enabled accounts. If your account is using a feature that isn't yet supported, the upgrade will not pass the validation step.

Review the [Blob Storage feature support in Azure Storage accounts](#) article to identify unsupported features. If you're using any of those unsupported features in your account, make sure to disable them before you begin the upgrade.

## Perform the upgrade

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

1. Sign in to the [Azure portal](#) to get started.
2. Locate your storage account and display the account overview.
3. Select **Data Lake Gen2 migration**.

The [Upgrade to a Storage account with Azure Data Lake Gen2 capabilities](#) configuration page appears.

Home > contoso

 contoso | Data Lake Gen2 migration ...  
Storage account

Search (Ctrl+ /)

Settings

-  Configuration
-  Data Lake Gen2 migration **Not started**
-  Resource sharing (CORS)
-  Advisor recommendations
-  Endpoints
-  Locks

Upgrade to a Storage account with Azure Data Lake Gen2 capabilities

If you're looking to use your storage account for data analytics and big data storage, you should consider upgrading to Azure Data Lake Storage Gen2, which will enable hierarchical namespace on the account. [Learn more](#)

Step 1: Review account changes before upgrading - Not started

Step 2: Validate account before upgrading - Not started

Step 3: Upgrade account - Not started

4. Expand the **Step 1: Review account changes before upgrading** section and click **Review and agree to changes**.
5. In the **Review account changes** page, select the checkbox and then click **Agree to changes**.
6. Expand the **Step 2: Validate account before upgrading** section and then click **Start validation**.

If validation fails, an error appears in the page. In some cases, a **View errors** link appears. If that link appears, select it.

#### Upgrade to a Storage account with Azure Data Lake Gen2 capabilities

If you're looking to use your storage account for data analytics and big data storage, you should consider upgrading to Azure Data Lake Storage Gen2, which will enable hierarchical namespace on the account. [Learn more](#)

- Step 1: Review account changes before upgrading - Completed
- Step 2: Validate account before upgrading - Failed
- All features that are not supported with ADLS Gen2 will be checked during validation, and a full list of discrepancies (if any) will be available as a blob in a generated container. This may take multiple attempts.
- Start validation**
-  Validation failed. Please fix the errors before trying again. [View errors](#)

Then, from the context menu of the **error.json** file, select **Download**.

The screenshot shows the Azure Storage Blob container 'hnsonerror'. On the left, there's a sidebar with 'Overview', 'Diagnose and solve problems', 'Access Control (IAM)', and 'Settings' sections. Under 'Settings', there are links for 'Shared access tokens', 'Access policy', 'Properties', 'Metadata', and 'Editor (preview)'. The main area displays a table with columns 'Name', 'Modified', and 'Access tier'. A single row is selected for 'error.json', which was modified on 9/1/2021 at 11:40:24 AM. A context menu is open over this row, showing options: 'View/edit', 'Download' (which is highlighted with a red box), and 'Properties'.

Open the downloaded file to determine why the account did not pass the validation step. If you have a Blob Storage feature that is fully supported, but which in Data Lake Storage Gen2 is supported only at the preview level or is not yet supported, validation might fail. To see how each Blob Storage feature is supported with Data Lake Storage Gen2, see [Blob Storage feature support in Azure Storage accounts](#).

The following JSON indicates that an incompatible feature is enabled on the account. In this case, you would disable the feature and then start the validation process again.

```
{
 "startTime": "2021-08-04T18:40:31.8465320Z",
 "id": "45c84a6d-6746-4142-8130-5ae9cf013a0",
 "incompatibleFeatures": [
 "Blob Delete Retention Enabled"
],
 "blobValidationErrors": [],
 "scannedBlobCount": 0,
 "invalidBlobCount": 0,
 "endTime": "2021-08-04T18:40:34.9371480Z"
}
```

- After your account has been successfully validated, expand the **Step 3: Upgrade account** section, and then click **Start upgrade**.

#### IMPORTANT

Write operations are disabled while your account is being upgraded. Read operations aren't disabled, but we strongly recommend that you suspend read operations as they might destabilize the upgrade process.

When the migration has completed successfully, a message similar to the following appears.

Home > contoso > contoso

# contoso | Data Lake Gen2 migration

Storage account

Search (Ctrl+ /)

- Settings
- Configuration
- Data Lake Gen2 migration
- Resource sharing (CORS)
- Advisor recommendations
- Endpoints
- Locks

Upgrade to a Storage account with Azure Data Lake Gen2 capabilities

If you're looking to use your storage account for data analytics and big data storage, you should consider upgrading to Azure Data Lake Storage Gen2, which will enable hierarchical namespace on the account. [Learn more](#)

This account has successfully been upgraded to an account with Azure Data...

Step 1: Review account changes before upgrading - Completed

Step 2: Validate account before upgrading - Completed

Step 3: Upgrade account - Completed

## Stop the upgrade

You can stop the migration before it completes.

- Portal
- PowerShell
- Azure CLI

To stop the upgrade before it completes, select **Cancel upgrade** while the upgrade is in progress.

Step 1: Review account changes before upgrading - Completed

Step 2: Validate account before upgrading - Completed

Step 3: Upgrade account - In progress

During the upgrade, the storage account will be offline. The upgrade may take several hours to complete. Once upgraded, an account cannot be reverted back.

Upgrade in progress (0% complete)

**Cancel upgrade**

## Migrate data, workloads, and applications

- Configure [services in your workloads](#) to point to either the **Blob service endpoint** or the **Data Lake storage endpoint**.

The screenshot shows the Microsoft Azure portal interface for a storage account named 'contoso'. The left sidebar has a 'Endpoints' section selected. The main content area displays two sets of endpoint information: 'Blob service' and 'Data Lake Storage'. The 'Blob service' endpoint is highlighted with a red box. The 'Data Lake Storage' endpoint is also highlighted with a red box.

2. For Hadoop workloads that use Windows Azure Storage Blob driver or [WASB](#) driver, make sure to modify them to use the [Azure Blob File System \(ABFS\)](#) driver. Unlike the WASB driver that makes requests to the **Blob service** endpoint, the ABFS driver will make requests to the **Data Lake Storage** endpoint of your account.
3. Test custom applications to ensure that they work as expected with your upgraded account.

[Multi-protocol access on Data Lake Storage](#) enables most applications to continue using Blob APIs without modification. If you encounter issues or you want to use APIs to work with directory operations and ACLs, consider moving some of your code to use Data Lake Storage Gen2 APIs. See guides for [.NET](#), [Java](#), [Python](#), [Node.js](#), and [REST](#).

4. Test any custom scripts to ensure that they work as expected with your upgraded account.

As is the case with Blob APIs, many of your scripts will likely work without requiring you to modify them. However, if needed, you can upgrade script files to use Data Lake Storage Gen2 [PowerShell cmdlets](#), and [Azure CLI commands](#).

## See also

[Introduction to Azure Data Lake storage Gen2](#)

# Manage blob containers using the Azure portal

8/22/2022 • 11 minutes to read • [Edit Online](#)

Azure Blob Storage allows you to store large amounts of unstructured object data. You can use Blob Storage to gather or expose media, content, or application data to users. Because all blob data is stored within containers, you must create a storage container before you can begin to upload data. To learn more about Blob Storage, read the [Introduction to Azure Blob storage](#).

In this how-to article, you learn how to work with container objects within the Azure portal.

## Prerequisites

To access Azure Storage, you'll need an Azure subscription. If you don't already have a subscription, create a [free account](#) before you begin.

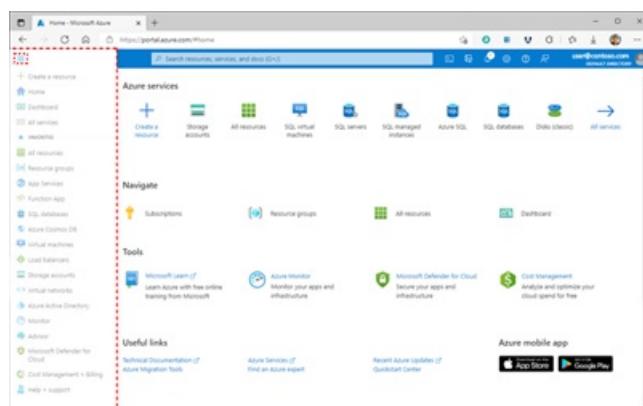
All access to Azure Storage takes place through a storage account. For this how-to article, create a storage account using the [Azure portal](#), Azure PowerShell, or Azure CLI. For help with creating a storage account, see [Create a storage account](#).

## Create a container

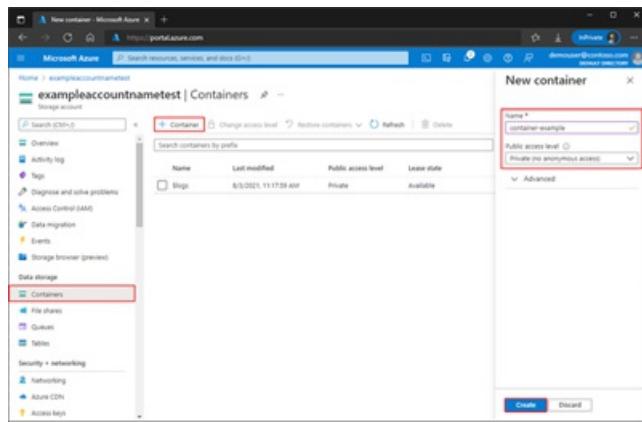
A container organizes a set of blobs, similar to a directory in a file system. A storage account can include an unlimited number of containers, and a container can store an unlimited number of blobs.

To create a container in the [Azure portal](#), follow these steps:

1. In the portal navigation pane on the left side of the screen, select **Storage accounts** and choose a storage account. If the navigation pane isn't visible, select the menu button to toggle its visibility.



2. In the navigation pane for the storage account, scroll to the **Data storage** section and select **Containers**.
3. Within the **Containers** pane, select the **+ Container** button to open the **New container** pane.
4. Within the **New Container** pane, provide a **Name** for your new container. The container name must be lowercase, must start with a letter or number, and can include only letters, numbers, and the dash (-) character. For more information about container and blob names, see [Naming and referencing containers, blobs, and metadata](#).
5. Set the **Public access level** for the container. The default level is **Private (no anonymous access)**. Read the article to learn how to [configure anonymous public read access for containers and blobs](#).
6. Select **Create** to create the container.



## Read container properties and metadata

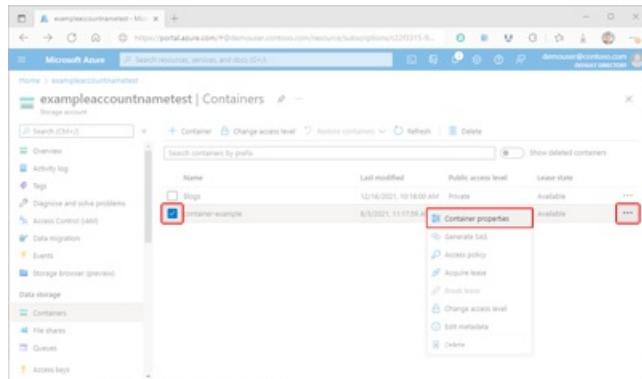
A container exposes both system properties and user-defined metadata. System properties exist on each Blob Storage resource. Some properties are read-only, while others can be read or set.

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.

### Container properties

To display the properties of a container within the [Azure portal](#), follow these steps:

1. Navigate to the list of containers within your storage account.
2. Select the checkbox next to the name of the container whose properties you want to view.
3. Select the container's **More** button (...), and select **Container properties** to display the container's **Properties** pane.



### Read and write container metadata

Users that have large numbers of objects within their storage account can organize their data logically within containers using metadata.

To manage a container's metadata within the [Azure portal](#), follow these steps:

1. Navigate to the list of containers in your storage account.
2. Select the checkbox next to the name of the container whose metadata you want to manage.
3. Select the container's **More** button (...), and then select **Edit metadata** to display the **Container metadata** pane.

4. The **Container metadata** pane will display existing metadata key-value pairs. Existing data can be edited by selecting an existing key or value and overwriting the data. You can add additional metadata by and supplying data in the empty fields provided. Finally, select **Save** to commit your data.

## Manage container and blob access

Properly managing access to containers and their blobs is key to ensuring that your data remains safe. The following sections illustrate ways in which you can meet your access requirements.

### Manage Azure RBAC role assignments for the container

Azure Active Directory (Azure AD) offers optimum security for Blob Storage resources. Azure role-based access control (Azure RBAC) determines what permissions a security principal has to a given resource. To grant access to a container, you'll assign an RBAC role at the container scope or above to a user, group, service principal, or managed identity. You may also choose to add one or more conditions to the role assignment.

You can read about the assignment of roles at [Assign Azure roles using the Azure portal](#).

### Enable anonymous public read access

Although anonymous read access for containers is supported, it's disabled by default. All access requests must require authorization until anonymous access is explicitly enabled. After anonymous access is enabled, any client will be able to read data within that container without authorizing the request.

Read about enabling public access level in the [Configure anonymous public read access for containers and blobs](#) article.

### Generate a shared access signature

A shared access signature (SAS) provides temporary, secure, delegated access to a client who wouldn't normally have permissions. A SAS gives you granular control over how a client can access your data. For example, you can specify which resources are available to the client. You can also limit the types of operations that the client can perform, and specify the duration.

Azure supports three types of SAS. A **service SAS** provides access to a resource in just one of the storage services: the Blob, Queue, Table, or File service. An **account SAS** is similar to a service SAS, but can permit access to resources in more than one storage service. A **user delegation SAS** is a SAS secured with Azure AD credentials and can only be used with Blob Storage service.

When you create a SAS, you may set access limitations based on permission level, IP address or range, or start

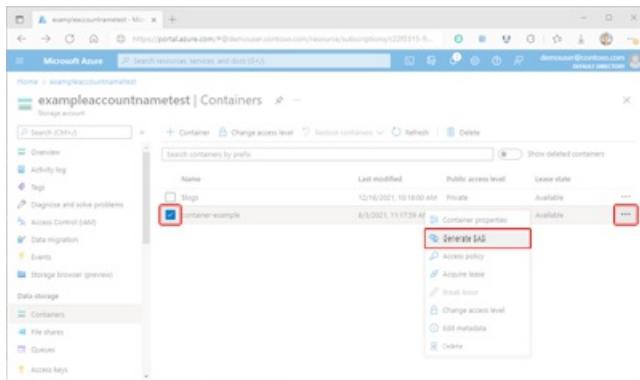
and expiry date and time. You can read more in [Grant limited access to Azure Storage resources using shared access signatures](#).

**Caution**

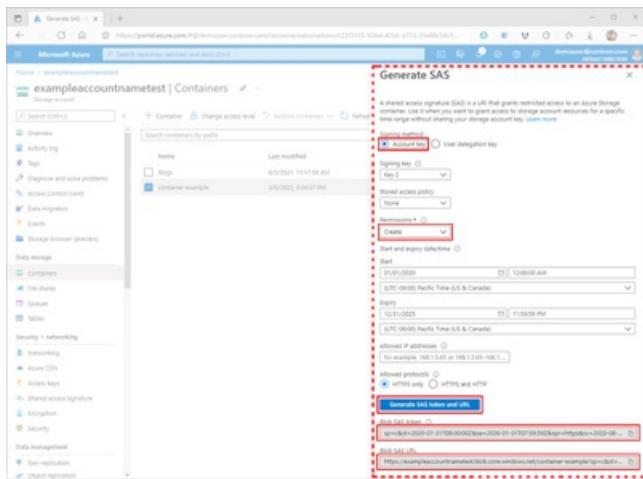
Any client that possesses a valid SAS can access data in your storage account as permitted by that SAS. It's important to protect a SAS from malicious or unintended use. Use discretion in distributing a SAS, and have a plan in place for revoking a compromised SAS.

To generate an SAS token using the [Azure portal](#), follow these steps:

1. In the Azure portal, navigate to the list of containers in your storage account.
2. Select the checkbox next to the name of the container for which you'll generate an SAS token.
3. Select the container's **More** button (...), and select **Generate SAS** to display the **Generate SAS** pane.



4. Within the **Generate SAS** pane, select the **Account key** value for the **Signing method** field.
5. In the **Signing method** field, select **Account key**. Choosing the account key will result in the creation of a service SAS.
6. In the **Signing key** field, select the desired key to be used to sign the SAS.
7. In the **Stored access policy** field, select **None**.
8. Select the **Permissions** field, then select the check boxes corresponding to the desired permissions.
9. In the **Start and expiry date/time** section, specify the desired **Start** and **Expiry** date, time, and time zone values.
10. Optionally, specify an IP address or a range of IP addresses from which to accept requests in the **Allowed IP addresses** field. If the request IP address doesn't match the IP address or address range specified on the SAS token, it won't be authorized.
11. Optionally, specify the protocol permitted for requests made with the SAS in the **Allowed protocols** field. The default value is **HTTPS**.
12. Review your settings for accuracy and then select **Generate SAS token and URL** to display the **Blob SAS token** and **Blob SAS URL** query strings.



- Copy and paste the blob SAS token and blob SAS url values in a secure location. They'll only be displayed once and can't be retrieved after the window is closed.

### Create a stored access or immutability policy

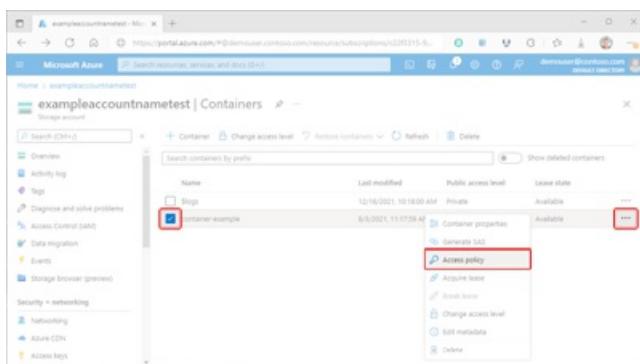
A **stored access policy** gives you additional server-side control over one or more shared access signatures. When you associate an SAS with a stored access policy, the SAS inherits the restrictions defined in the policy. These extra restrictions allow you to change the start time, expiry time, or permissions for a signature. You can also revoke it after it has been issued.

**Immutability policies** can be used to protect your data from overwrites and deletes. Immutability policies allow objects to be created and read, but prevents their modification or deletion for a specific duration. Blob Storage supports two types of immutability policies. A **time-based retention policy** prohibits write and delete operations for a defined period of time. A **legal hold** also prohibits write and delete operations, but must be explicitly cleared before those operations can resume.

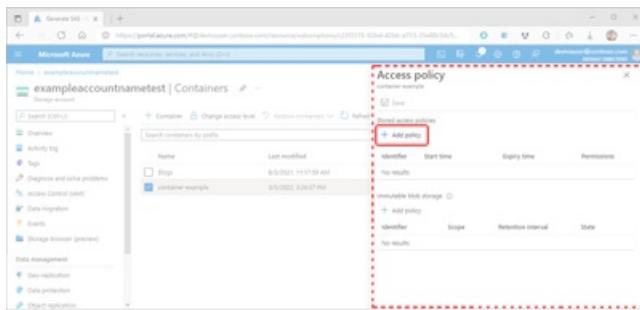
### Create a stored access policy

Configuring a stored access policy is a two-step process: the policy must first be defined, and then applied to the container afterward. To configure a stored access policy, follow these steps:

- In the Azure portal, navigate to the list of containers in your storage account.
- Select the checkbox next to the name of the container for which you'll generate an SAS token.
- Select the container's **More** button (...), and select **Access policy** to display the **Access policy** pane.



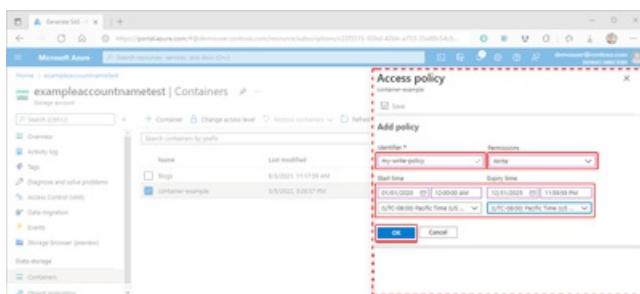
- Within the **Access policy** pane, select **+ Add policy** in the **Stored access policies** section to display the **Add policy** pane. Any existing policies will be displayed in either the appropriate section.



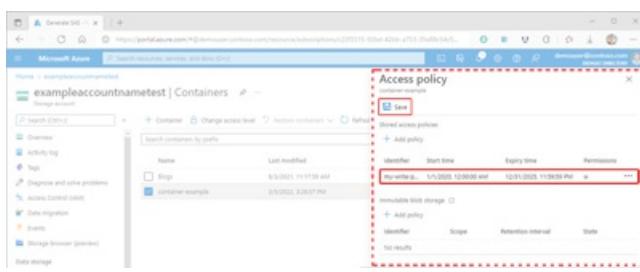
5. Within the **Add policy** pane, select the **Identifier** box and add a name for your new policy.
6. Select the **Permissions** field, then select the check boxes corresponding to the permissions desired for your new policy.
7. Optionally, provide date, time, and time zone values for **Start time** and **Expiry time** fields to set the policy's validity period.
8. Review your settings for accuracy and then select **OK** to update the **Access policy** pane.

**Caution**

Although your policy is now displayed in the **Stored access policy** table, it is still not applied to the container. If you navigate away from the **Access policy** pane at this point, the policy will *not* be saved or applied and you will lose your work.



9. In the **Access policy** pane, select **+ Add policy** to define another policy, or select **Save** to apply your new policy to the container. After creating at least one stored access policy, you'll be able to associate other secure access signatures (SAS) with it.



#### Create an immutability policy

Read more about how to [Configure immutability policies for containers](#). For help with implementing immutability policies, follow the steps outlined in the [Configure a retention policy](#) or [Configure or clear a legal hold](#) articles.

## Manage leases

A container lease is used to establish or manage a lock for delete operations. When a lease is acquired within the Azure portal, the lock can only be created with an infinite duration. When created programmatically, the lock duration can range from 15 to 60 seconds, or it can be infinite.

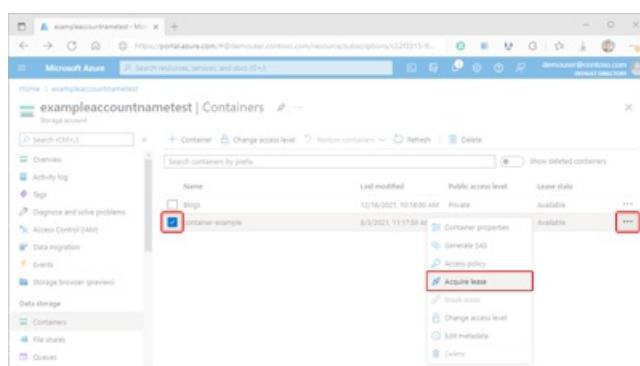
There are five different lease operation modes, though only two are available within the Azure portal:

	USE CASE	AVAILABLE IN AZURE PORTAL
Acquire mode	Request a new lease.	✓
Renew mode	Renew an existing lease.	
Change mode	Change the ID of an existing lease.	
Release mode	End the current lease; allows other clients to acquire a new lease	✓
Break mode	End the current lease; prevents other clients from acquiring a new lease during the current lease period	

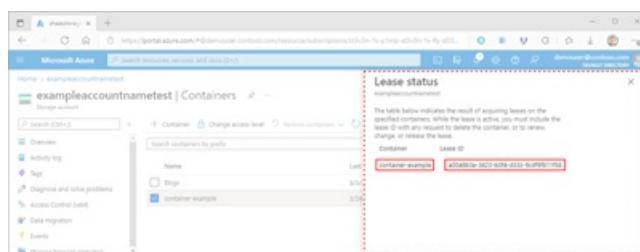
## Acquire a lease

To acquire a lease using the Azure portal, follow these steps:

1. In the Azure portal, navigate to the list of containers in your storage account.
2. Select the checkbox next to the name of the container for which you'll acquire a lease.
3. Select the container's **More** button (...), and select **Acquire lease** to request a new lease and display the details in the **Lease status** pane.



4. The **Container** and **Lease ID** property values of the newly requested lease are displayed within the **Lease status** pane. Copy and paste these values in a secure location. They'll only be displayed once and can't be retrieved after the pane is closed.



## Break a lease

To break a lease using the Azure portal, follow these steps:

1. In the Azure portal, navigate to the list of containers in your storage account.
2. Select the checkbox next to the name of the container for which you'll break a lease.
3. Select the container's **More** button (...), and select **Break lease** to break the lease.

The screenshot shows the Azure portal interface for a storage account named 'exampleaccountnametest'. In the left sidebar, under 'Storage accounts', 'Containers' is selected. The main area displays a table of containers. One container, 'container-example', is selected and highlighted with a blue box. A context menu is open for this container, with the 'Break lease' option highlighted by a red box.

4. After the lease is broken, the selected container's **Lease state** value will update, and a status confirmation will appear.

The screenshot shows the Azure portal interface for the same storage account. The 'Containers' section is selected. The 'Lease state' column for 'container-example' now shows 'broken'. A toast notification at the top right of the screen indicates 'Successfully broke lease on container(s)'. The 'Break lease' option in the context menu is no longer highlighted.

## Delete containers

When you delete a container within the Azure portal, all blobs within the container will also be deleted.

### WARNING

Following the steps below may permanently delete containers and any blobs within them. Microsoft recommends enabling container soft delete to protect containers and blobs from accidental deletion. For more info, see [Soft delete for containers](#).

To delete a container within the [Azure portal](#), follow these steps:

1. In the Azure portal, navigate to the list of containers in your storage account.
2. Select the container to delete.
3. Select the **More** button (...), and select **Delete**.

The screenshot shows the Azure portal interface for the storage account. The 'Containers' section is selected. A container named 'container-example' is selected and highlighted with a blue box. A context menu is open for this container, with the 'Delete' option highlighted by a red box.

4. In the **Delete container(s)** dialog, confirm that you want to delete the container.

In some cases, it's possible to retrieve containers that have been deleted. If soft delete data protection option is enabled on your storage account, you can access containers deleted within the associated retention period. To

learn more about soft delete, refer to the [Soft delete for containers](#) article.

## View soft-deleted containers

When soft delete is enabled, you can view soft-deleted containers within the Azure portal. Soft-deleted containers are visible during the specified retention period. After the retention period expires, a soft-deleted container is permanently deleted and is no longer visible.

To view soft-deleted containers within the [Azure portal](#), follow these steps:

1. Navigate to your storage account within the Azure portal and view the list of your containers.
2. Toggle the **Show deleted containers** switch to include deleted containers in the list.

The screenshot shows the 'Containers' blade in the Azure Storage account 'softdeletesamples'. The left sidebar lists Blob service and File service options. Under Blob service, 'Containers' is selected, showing a list of six containers: sample-container-1 through sample-container-6. Container-2 and Container-3 are marked as 'Deleted'. A red box highlights the 'Show deleted containers' toggle switch at the top right of the list. The table columns are Name, Status, Last modified, Public access level, and Lease state.

Name	Status	Last modified	Public access level	Lease state
sample-container-1	Active	7/21/2020, 3:23:55 PM	Private	Available
sample-container-2	Deleted	7/21/2020, 3:23:55 PM	Private	-
sample-container-3	Deleted	7/21/2020, 3:23:55 PM	Private	-
sample-container-4	Active	7/21/2020, 3:23:55 PM	Private	Available
sample-container-5	Active	7/21/2020, 3:23:55 PM	Private	Available
sample-container-6	Active	7/21/2020, 3:23:55 PM	Private	Available

## Restore a soft-deleted container

You can restore a soft-deleted container and its contents within the retention period. To restore a soft-deleted container within the [Azure portal](#), follow these steps:

1. Navigate to your storage account within the Azure portal and view the list of your containers.
2. Display the context menu for the container you wish to restore, and choose **Undelete** from the menu.

The screenshot shows the same 'Containers' blade as before, but with a different context menu for 'sample-container-2'. The menu items are 'Container properties' (with a gear icon) and 'Undelete' (with a circular arrow icon). A blue dashed box highlights the 'Undelete' option. The rest of the interface is identical to the previous screenshot.

## See also

- [Create a storage account](#)
- [Manage blob containers using PowerShell](#)

# Manage blob containers using Azure CLI

8/22/2022 • 11 minutes to read • [Edit Online](#)

Azure blob storage allows you to store large amounts of unstructured object data. You can use blob storage to gather or expose media, content, or application data to users. Because all blob data is stored within containers, you must create a storage container before you can begin to upload data. To learn more about blob storage, read the [Introduction to Azure Blob storage](#).

The Azure CLI is Azure's cross-platform command-line experience for managing Azure resources. You can use it in your browser with Azure Cloud Shell. You can also install it on macOS, Linux, or Windows and run it locally from the command line.

In this how-to article, you learn to use the Azure CLI with Bash to work with container objects.

## Prerequisites

To access Azure Storage, you'll need an Azure subscription. If you don't already have a subscription, create a [free account](#) before you begin.

All access to Azure Storage takes place through a storage account. For this quickstart, create a storage account using the [Azure portal](#), Azure PowerShell, or Azure CLI. For help creating a storage account, see [Create a storage account](#).

### Prepare your environment for the Azure CLI

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Azure Cloud Shell Quickstart - Bash](#).  
[Launch Cloud Shell](#)
- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see [How to run the Azure CLI in a Docker container](#).
  - If you're using a local installation, sign in to the Azure CLI by using the `az login` command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run `az version` to find the version and dependent libraries that are installed. To upgrade to the latest version, run `az upgrade`.
- It's always a good idea to install the latest version of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

### Authorize access to Blob storage

You can authorize access to Blob storage from the Azure CLI either with Azure AD credentials or by using the storage account access key. Using Azure AD credentials is recommended, and this article's examples use Azure AD exclusively.

Azure CLI commands for data operations against Blob storage support the `--auth-mode` parameter, which enables you to specify how to authorize a given operation. Set the `--auth-mode` parameter to `login` to

authorize with Azure AD credentials. For more information, see [Authorize access to blob or queue data with Azure CLI](#).

Run the `login` command to open a browser and connect to your Azure subscription.

```
az login
```

## Create a container

To create a container with Azure CLI, call the `az storage container create` command. The following example illustrates three options for the creation of blob containers with the `az storage container create` command. The first approach creates a single container, while the remaining two approaches use Bash scripting operations to automate container creation.

To use this example, supply values for the variables and ensure that you've logged in. Remember to replace the placeholder values in brackets with your own values.

```
#!/bin/bash
storageAccount=<storage-account>
containerName="demo-container-1"
containerPrefix="demo-container-"

Approach 1: Create a container
az storage container create \
 --name $containerName \
 --account-name $storageAccount \
 --auth-mode login

Approach 2: Create containers with a loop
for value in {2..5}
do
 az storage container create \
 --name ${containerPrefix}${value} \
 --account-name $storageAccount \
 --auth-mode login
done

Approach 3: Create containers by splitting multiple values
containerList="${containerPrefix}6 ${containerPrefix}7 ${containerPrefix}8"
for container in $containerList
do
 az storage container create \
 --name $container \
 --account-name $storageAccount \
 --auth-mode login
done
```

## List containers

Use the `az storage container list` command to retrieve a list of storage containers. To return a list of containers whose names begin with a given character string, pass the string as the `--prefix` parameter value.

The `--num-results` parameter can be used to limit the number of containers returned by the request. Azure Storage limits the number of containers returned by a single listing operation to 5000. This limit ensures that manageable amounts of data are retrieved. If the number of containers returned exceeds either the `--num-results` value or the service limit, a continuation token is returned. This token allows you to use multiple requests to retrieve any number of containers.

You can also use the `--query` parameter to execute a [JMESPath query](#) on the results of commands. JMESPath is

a query language for JSON that allows you to select and modify data returned from CLI output. Queries are executed on the JSON output before it can be formatted. For more information, see [How to query Azure CLI command output using a JMESPath query](#).

The following example first lists the maximum number of containers (subject to the service limit). Next, it lists three containers whose names begin with the prefix *container-* by supplying values for the `--num-results` and `--prefix` parameters. Finally, a single container is listed by supplying a known container name to the `--prefix` parameter.

Read more about the [az storage container list](#).

```
#!/bin/bash
storageAccount=<storage-account>
containerPrefix="demo-container-"
containerName="demo-container-1"
numResults="3"

Approach 1: List maximum containers
az storage container list \
--account-name $storageAccount \
--auth-mode login

Approach 2: List a defined number of named containers
az storage container list \
--prefix $containerPrefix \
--num-results $numResults \
--account-name $storageAccount \
--auth-mode login

Approach 3: List an individual container
az storage container list \
--prefix $containerPrefix \
--query "[?name=='$containerName']" \
--account-name $storageAccount \
--auth-mode login
```

## Read container properties and metadata

A container exposes both system properties and user-defined metadata. System properties exist on each blob storage resource. Some properties are read-only, while others can be read or set. Under the covers, some system properties map to certain standard HTTP headers.

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.

### Container properties

To display the properties of a container with Azure CLI, call the [az storage container show](#) command.

In the following example, the first approach displays the properties of a single named container. Afterward, it retrieves all containers with the **demo-container-** prefix and iterates through them, listing their properties. Remember to replace the placeholder values with your own values.

```

#!/bin/bash
storageAccount=<storage-account>
containerPrefix="demo-container-"
containerName="demo-container-1"

Show a named container's properties
az storage container show \
 --name $containerName \
 --account-name $storageAccount \
 --auth-mode login

List several containers and show their properties
containerList=$(az storage container list \
 --query "[].name" \
 --prefix $containerPrefix \
 --account-name $storageAccount \
 --auth-mode login \
 --output tsv)

for row in $containerList
do
 tmpRow=$(echo $row | sed -e 's/\r//g')
 az storage container show --name $tmpRow --account-name $storageAccount --auth-mode login
done

```

## Read and write container metadata

Users that have many thousands of objects within their storage account can quickly locate specific containers based on their metadata. To read the metadata, you'll use the `az storage container metadata show` command. To update metadata, you'll need to call the `az storage container metadata update` command. The method only accepts space-separated key-value pairs. For more information, see the [az storage container metadata](#) documentation.

The first example below updates and then retrieves a named container's metadata. The second example iterates the list of containers matching the `-prefix` value. Containers with names containing even numbers have their metadata set with values contained in the *metadata* variable.

```

#!/bin/bash
storageAccount=<storage-account>
containerName="demo-container-1"
containerPrefix="demo-container-"

Create metadata string
metadata="key=value pie=delicious"

Update named container metadata
az storage container metadata update \
 --name $containerName \
 --metadata $metadata \
 --account-name $storageAccount \
 --auth-mode login

Display metadata
az storage container metadata show \
 --name $containerName \
 --account-name $storageAccount \
 --auth-mode login

Get list of containers
containerList=$(az storage container list \
 --query "[].name" \
 --prefix $containerPrefix \
 --account-name $storageAccount \
 --auth-mode login \
 --output tsv)

Update and display metadata
for row in $containerList
do
 #Get the container's number
 tmpName=$(echo $row | sed -e 's/\r//g')
 if [`expr ${tmpName: ${#containerPrefix}} % 2` == 0]
 then
 az storage container metadata update \
 --name $tmpName \
 --metadata $metadata \
 --account-name $storageAccount \
 --auth-mode login

 echo $tmpName

 az storage container metadata show \
 --name $tmpName \
 --account-name $storageAccount \
 --auth-mode login
 fi
done

```

## Delete containers

Depending on your use case, you can delete a single container or a group of containers with the `az storage container delete` command. When deleting a list of containers, you'll need to use conditional operations as shown in the examples below.

### WARNING

Running the following examples may permanently delete containers and blobs. Microsoft recommends enabling container soft delete to protect containers and blobs from accidental deletion. For more info, see [Soft delete for containers](#).

```

#!/bin/bash
storageAccount="<storage-account>"
containerName="demo-container-1"
containerPrefix="demo-container-"

Delete a single named container
az storage container delete \
--name $containerName \
--account-name $storageAccount \
--auth-mode login

Delete containers by iterating a loop
list=$(az storage container list \
--query "[].name" \
--prefix $containerPrefix \
--account-name $storageAccount \
--auth-mode login \
--output tsv)
for row in $list
do
 tmpName=$(echo $row | sed -e 's/\r//g')
 az storage container delete \
 --name $tmpName \
 --account-name $storageAccount \
 --auth-mode login
done

```

If you have container soft delete enabled for your storage account, then it's possible to retrieve containers that have been deleted. If your storage account's soft delete data protection option is enabled, the `--include-deleted` parameter will return containers deleted within the associated retention period. The `--include-deleted` parameter can only be used to return containers when used with the `--prefix` parameter. To learn more about soft delete, refer to the [Soft delete for containers](#) article.

Use the following example to retrieve a list of containers deleted within the storage account's associated retention period.

```

#!/bin/bash
storageAccount="<storage-account>"
containerPrefix="demo-container-"

Retrieve a list of containers including those recently deleted
az storage container list \
--prefix $containerPrefix \
--include-deleted \
--account-name $storageAccount \
--auth-mode login

```

## Restore a soft-deleted container

As mentioned in the [List containers](#) section, you can configure the soft delete data protection option on your storage account. When enabled, it's possible to restore containers deleted within the associated retention period. Before you can follow this example, you'll need to enable soft delete and configure it on at least one of your storage accounts.

The following examples explain how to restore a soft-deleted container with the `az storage container restore` command. You'll need to supply values for the `--name` and `--version` parameters to ensure that the correct version of the container is restored. If you don't know the version number, you can use the `az storage container list` command to retrieve it as shown in the first example. The second example finds and restores all deleted containers within a specific storage account.

To learn more about the soft delete data protection option, refer to the [Soft delete for containers](#) article.

```
#!/bin/bash
storageAccount=<storage-account>
containerName="demo-container-1"

Restore an individual named container
containerVersion=$(az storage container list \
 --account-name $storageAccount \
 --query "[?name=='$containerName'].[version]" \
 --auth-mode login \
 --output tsv \
 --include-deleted | sed -e 's/\r//g')

az storage container restore \
 --name $containerName \
 --deleted-version $containerVersion \
 --account-name $storageAccount \
 --auth-mode login

Restore a list of deleted containers
containerList=$(az storage container list \
 --account-name $storageAccount \
 --include-deleted \
 --auth-mode login \
 --query "[?deleted].{name:name,version:version}" \
 -o json)

for row in $(echo "${containerList}" | jq -c '.[]')
do
 tmpName=$(echo $row | jq -r '.name')
 tmpVersion=$(echo $row | jq -r '.version')
 az storage container restore \
 --account-name $storageAccount \
 --name $tmpName \
 --deleted-version $tmpVersion \
 --auth-mode login
done
```

## Get a shared access signature for a container

A shared access signature (SAS) provides delegated access to Azure resources. A SAS gives you granular control over how a client can access your data. For example, you can specify which resources are available to the client. You can also limit the types of operations that the client can perform, and specify the interval over which the SAS is valid.

A SAS is commonly used to provide temporary and secure access to a client who wouldn't normally have permissions. To generate either a service or account SAS, you'll need to supply values for the `--account-name` and `--account-key` parameters. An example of this scenario would be a service that allows users read and write their own data to your storage account.

Azure Storage supports three types of shared access signatures: user delegation, service, and account SAS. For more information on shared access signatures, see the [Grant limited access to Azure Storage resources using shared access signatures](#) article.

### Caution

Any client that possesses a valid SAS can access data in your storage account as permitted by that SAS. It's important to protect a SAS from malicious or unintended use. Use discretion in distributing a SAS, and have a plan in place for revoking a compromised SAS.

The following example illustrates the process of configuring a service SAS for a specific container using the `az storage container generate-sas` command. Because it's generating a service SAS, the example first retrieves

the storage account key to pass as the `--account-key` value.

The example will configure the SAS with start and expiry times and a protocol. It will also specify the **delete**, **read**, **write**, and **list** permissions in the SAS using the `--permissions` parameter. You can reference the full table of permissions in the [Create a service SAS](#) article.

Copy and paste the Blob SAS token value in a secure location. It will only be displayed once and can't be retrieved once Bash is closed. To construct the SAS URL, append the SAS token (URI) to the URL for the storage service.

```
#!/bin/bash
storageAccount=<storage-account>
containerName="demo-container-1"
permissions="drwl"
expiry=`date -u -d "30 minutes" '+%Y-%m-%dT%H:%MZ'`

accountKey=$(az storage account keys list \
 --account-name $storageAccount \
 --query "[?permissions == 'FULL'].[value]" \
 --output tsv)

accountKey=$(echo $accountKey | cut -d' ' -f1)

az storage container generate-sas \
 --name $containerName \
 --https-only \
 --permissions dlrw \
 --expiry $expiry \
 --account-key $accountKey \
 --account-name $storageAccount
```

## Next steps

In this how-to article, you learned how to manage containers in Azure blob storage. To learn more about working with blob storage by using Azure CLI, select an option below.

[Manage block blobs with Azure CLI](#)

[Azure CLI samples for Blob storage](#)

# Manage blob containers using PowerShell

8/22/2022 • 9 minutes to read • [Edit Online](#)

Azure blob storage allows you to store large amounts of unstructured object data. You can use blob storage to gather or expose media, content, or application data to users. Because all blob data is stored within containers, you must create a storage container before you can begin to upload data. To learn more about blob storage, read the [Introduction to Azure Blob storage](#).

This how-to article explains how to work with both individual and multiple storage container objects.

## Prerequisites

- An Azure subscription. See [Get Azure free trial](#).
- Azure PowerShell module Az, which is the recommended PowerShell module for interacting with Azure.  
To get started with the Az PowerShell module, see [Install Azure PowerShell](#).

You'll need to obtain authorization to an Azure subscription before you can use the examples in this article. Authorization can occur by authenticating with an Azure Active Directory (Azure AD) account or using a shared key. The examples in this article use Azure AD authentication in conjunction with context objects. Context objects encapsulate your Azure AD credentials and pass them on subsequent data operations, eliminating the need to reauthenticate.

To sign in to your Azure account with an Azure AD account, open PowerShell and call the [Connect-AzAccount](#) cmdlet.

```
Connect to your Azure subscription
Connect-AzAccount
```

After the connection has been established, create the storage account context by calling the [New-AzStorageContext](#) cmdlet. Include the `-UseConnectedAccount` parameter so that data operations will be performed using your Azure AD credentials.

```
Create a context object using Azure AD credentials
$ctx = New-AzStorageContext -StorageAccountName <storage account name> -UseConnectedAccount
```

Remember to replace the placeholder values in brackets with your own values. For more information about signing into Azure with PowerShell, see [Sign in with Azure PowerShell](#).

## Create a container

To create containers with PowerShell, call the [New-AzStorageContainer](#) cmdlet. There are no limits to the number of blobs or containers that can be created within a storage account. Containers cannot be nested within other containers.

The following example illustrates three options for the creation of blob containers with the [New-AzStorageContainer](#) cmdlet. The first approach creates a single container, while the remaining two approaches leverage PowerShell operations to automate container creation.

To use this example, supply values for the variables and ensure that you've created a connection to your Azure subscription. Remember to replace the placeholder values in brackets with your own values.

```

Create variables
$containerName = "individual-container"
$prefixName = "loop"

Approach 1: Create a container
New-AzStorageContainer -Name $containerName -Context $ctx

Approach 2: Create containers with a PowerShell loop
for ($i = 1; $i -le 3; $i++) {
 New-AzStorageContainer -Name (-join($prefixName, $i)) -Context $ctx
}

Approach 3: Create containers using the PowerShell Split method
"$(($prefixName)4 $($prefixName)5 $($prefixName)6".split() | New-AzStorageContainer -Context $ctx

```

The result provides the name of the storage account and confirms the creation of the new container.

Storage Account Name: demostorageaccount		
Name	PublicAccess	LastModified
individual-container	Off	11/2/2021 4:09:05 AM +00:00
loop-container1	Off	11/2/2021 4:09:05 AM +00:00
loop-container2	Off	11/2/2021 4:09:05 AM +00:00
loop-container3	Off	11/2/2021 4:09:05 AM +00:00
loop-container4	Off	11/2/2021 4:09:05 AM +00:00
loop-container5	Off	11/2/2021 4:09:05 AM +00:00
loop-container6	Off	11/2/2021 4:09:05 AM +00:00

## List containers

Use the `Get-AzStorageContainer` cmdlet to retrieve storage containers. To retrieve a single container, include the `-Name` parameter. To return a list of containers that begins with a given character string, specify a value for the `-Prefix` parameter.

The following example retrieves both an individual container and a list of container resources.

```

Create variables
$containerName = "individual-container"
$prefixName = "loop-"

Approach 1: Retrieve an individual container
Get-AzStorageContainer -Name $containerName -Context $ctx
Write-Host

Approach 2: Retrieve a list of containers
Get-AzStorageContainer -Prefix $prefixName -Context $ctx

```

The result provides the URI of the blob endpoint and lists the containers retrieved by name and prefix.

```
Storage Account Name: demostorageaccount
```

Name	PublicAccess	LastModified	IsDeleted	VersionId
individual-container		11/2/2021 5:52:08 PM +00:00		
loop-container1		11/2/2021 12:22:00 AM +00:00		
loop-container2		11/2/2021 12:22:00 AM +00:00		
loop-container1		11/2/2021 12:22:00 AM +00:00		
loop-container2		11/2/2021 12:22:00 AM +00:00		
loop-container3		11/2/2021 12:22:00 AM +00:00	True	01D7E7129FDBD7D4
loop-container4		11/2/2021 12:22:00 AM +00:00	True	01D7E8A5EF01C787

## Read container properties and metadata

A container exposes both system properties and user-defined metadata. System properties exist on each blob storage resource. Some properties are read-only, while others can be read or set. Under the covers, some system properties map to certain standard HTTP headers.

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.

### Container properties

The following example retrieves all containers with the **demo** prefix and iterates through them, listing their properties.

```
Create variable
$prefix = "loop"

Get containers
$containers = Get-AzStorageContainer -Prefix $prefix -Context $ctx

Iterate containers, display properties
Foreach ($container in $containers)
{
 $containerProperties = $container.BlobContainerClient.GetProperties()
 Write-Host $container.Name "properties:"
 $containerProperties.Value
}
```

The results display all containers with the prefix **loop** and list their properties.

```
loop-container1 properties:

LastModified : 12/7/2021 7:47:17 PM +00:00
LeaseStatus : Unlocked
LeaseState : Available
LeaseDuration : Infinite
PublicAccess :
HasImmutabilityPolicy : False
HasLegalHold : False
DefaultEncryptionScope : $account-encryption-key
PreventEncryptionScopeOverride : False
DeletedOn :
RemainingRetentionDays :
ETag : 0x8D9B9BA602806DA
Metadata : {}
HasImmutableStorageWithVersioning : False

loop-container2 properties:
LastModified : 12/7/2021 7:47:18 PM +00:00
LeaseStatus : Unlocked
LeaseState : Available
LeaseDuration : Infinite
PublicAccess :
HasImmutabilityPolicy : False
HasLegalHold : False
DefaultEncryptionScope : $account-encryption-key
PreventEncryptionScopeOverride : False
DeletedOn :
RemainingRetentionDays :
ETag : 0x8D9B9BA605996AE
Metadata : {}
HasImmutableStorageWithVersioning : False
```

## Read and write container metadata

Users that have many thousands of objects within their storage account can quickly locate specific containers based on their metadata. To access the metadata, you'll use the `BlobContainerClient` object. This object allows you to access and manipulate containers and their blobs. To update metadata, you'll need to call the `SetMetadata()` method. The method only accepts key-value pairs stored in a generic `IDictionary` object. For more information, see the [BlobContainerClient class](#)

The example below first updates a container's metadata and afterward retrieve a container's metadata. The example flushes the sample container from memory and retrieves it again to ensure that metadata isn't being read from the object in memory.

```

Create variable
$containerName = "individual-container"

Retrieve container
$container = Get-AzStorageContainer -Name $containerName -Context $ctx

Create IDictionary, add key-value metadata pairs to IDictionary
$metadata = New-Object System.Collections.Generic.Dictionary[String, String]
$metadata.Add("CustomerName", "Anthony Bennedetto")
$metadata.Add("CustomerDOB", "08/03/1926")
$metadata.Add("CustomerBirthplace", "Long Island City")

Update metadata
$container.BlobContainerClient.SetMetadata($metadata, $null)

Flush container from memory, retrieve updated container
$container = $null
$container = Get-AzStorageContainer -Name $containerName -Context $ctx

Display metadata
$properties = $container.BlobContainerClient.GetProperties()
Write-Host $container.Name "metadata:"
Write-Host $properties.Value.Metadata

```

The results display the complete metadata for a container.

```

individual-container metadata:

[CustomerName, Anthony Bennedetto] [CustomerDOB, 08/03/1926] [CustomerBirthplace, Long Island City]

```

## Get a shared access signature for a container

A shared access signature (SAS) provides delegated access to Azure resources. A SAS gives you granular control over how a client can access your data. For example, you can specify which resources are available to the client. You can also limit the types of operations that the client can perform, and specify the amount of time for which the actions can be taken.

A SAS is commonly used to provide temporary and secure access to a client who wouldn't normally have permissions. An example of this scenario would be a service that allows users read and write their own data to your storage account.

Azure Storage supports three types of shared access signatures: user delegation, service, and account SAS. For more information on shared access signatures, see the [Create a service SAS for a container or blob](#) article.

**Caution**

Any client that possesses a valid SAS can access data in your storage account as permitted by that SAS. It's important to protect a SAS from malicious or unintended use. Use discretion in distributing a SAS, and have a plan in place for revoking a compromised SAS.

The following example illustrates the process of configuring a service SAS for a specific container using the `New-AzStorageContainerSASToken` cmdlet. The example will configure the SAS with start and expiry times and a protocol. It will also specify the `read`, `write`, and `list` permissions in the SAS using the `-Permission` parameter. You can reference the full table of permissions in the [Create a service SAS](#) article.

```

Create variables
$accountName = "<storage-account>"
$containerName = "individual-container"
$startTime = Get-Date
$expiryTime = $startTime.AddDays(7)
$permissions = "rwl"
$protocol = "HttpsOnly"

Create a context object using Azure AD credentials, retrieve container
$ctx = New-AzStorageContext -StorageAccountName $accountName -UseConnectedAccount

Approach 1: Generate SAS token for a specific container
$sas = New-AzStorageContainerSASToken `

-Context $ctx `

-Name $containerName `

-StartTime $startTime `

-ExpiryTime $expiryTime `

-Permission $permissions `

-Protocol $protocol

Approach 2: Generate SAS tokens for a container list using pipeline
Get-AzStorageContainer -Container $filterName -Context $ctx | New-AzStorageContainerSASToken `

-Context $ctx `

-StartTime $startTime `

-ExpiryTime $expiryTime `

-Permission $permissions `

-Protocol $protocol | Write-Output

```

## Delete containers

Depending on your use case, you can delete a container or list of containers with the `Remove-AzStorageContainer` cmdlet. When deleting a list of containers, you can leverage conditional operations, loops, or the PowerShell pipeline as shown in the examples below.

```

Create variables
$accountName = "<storage-account>"
$containerName = "individual-container"
$prefixName = "loop-"

Delete a single named container
Remove-AzStorageContainer -Name $containerName -Context $ctx

Iterate a loop, deleting containers
for ($i = 1; $i -le 2; $i++) {
 Remove-AzStorageContainer -Name (-join($containerPrefix, $i)) -Context $ctx
}

Retrieve container list, delete using a pipeline
Get-AzStorageContainer -Prefix $prefixName -Context $ctx | Remove-AzStorageContainer

```

In some cases, it's possible to retrieve containers that have been deleted. If your storage account's soft delete data protection option is enabled, the `-IncludeDeleted` parameter will return containers deleted within the associated retention period. The `-IncludeDeleted` parameter can only be used in conjunction with the `-Prefix` parameter when returning a list of containers. To learn more about soft delete, refer to the [Soft delete for containers](#) article.

Use the following example to retrieve a list of containers deleted within the storage account's associated retention period.

```
Retrieve a list of containers including those recently deleted
Get-AzStorageContainer -Prefix $prefixName -Context $ctx -IncludeDeleted
```

## Restore a soft-deleted container

As mentioned in the [List containers](#) section, you can configure the soft delete data protection option on your storage account. When enabled, it's possible to restore containers deleted within the associated retention period.

The following example explains how to restore a soft-deleted container with the `Restore-AzStorageContainer` cmdlet. Before you can follow this example, you'll need to enable soft delete and configure it on at least one of your storage accounts.

To learn more about the soft delete data protection option, refer to the [Soft delete for containers](#) article.

```
Create variables
$accountName = "<storage-account>"
$prefixName = "loop-"

Create a context object using Azure AD credentials
$ctx = New-AzStorageContext -StorageAccountName $accountName -UseConnectedAccount

Retrieve all containers, filter deleted containers, restore deleted containers
Get-AzStorageContainer -Prefix $prefixName -IncludeDeleted -Context $ctx | ? { $_.IsDeleted } | Restore-AzStorageContainer
```

The results display all containers with the prefix **demo** which have been restored.

Storage Account Name: demostorageaccount				
Name	PublicAccess	LastModified	IsDeleted	VersionId
loop-container3	-----	-----	-----	-----
loop-container4	-----	-----	-----	-----

## See also

- [Run PowerShell commands with Azure AD credentials to access blob data](#)
- [Create a storage account](#)

# Manage block blobs with Azure CLI

8/22/2022 • 19 minutes to read • [Edit Online](#)

Blob storage supports block blobs, append blobs, and page blobs. Block blobs are optimized for uploading large amounts of data efficiently. Block blobs are ideal for storing images, documents, and other types of data not subjected to random read and write operations. This article explains how to work with block blobs.

## Prerequisites

To access Azure Storage, you'll need an Azure subscription. If you don't already have a subscription, create a [free account](#) before you begin.

All access to Azure Storage takes place through a storage account. For this quickstart, create a storage account using the [Azure portal](#), Azure PowerShell, or Azure CLI. For help creating a storage account, see [Create a storage account](#).

### Prepare your environment for the Azure CLI

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Azure Cloud Shell Quickstart - Bash](#).

 [Launch Cloud Shell](#)

- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see [How to run the Azure CLI in a Docker container](#).
  - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).
- This article requires version 2.0.46 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

### Authorize access to Blob storage

You can authorize access to Blob storage from the Azure CLI either with Azure AD credentials or by using a storage account access key. Using Azure AD credentials is recommended, and this article's examples use Azure AD exclusively.

Azure CLI commands for data operations against Blob storage support the `--auth-mode` parameter, which enables you to specify how to authorize a given operation. Set the `--auth-mode` parameter to `login` to authorize with Azure AD credentials. Only Blob storage data operations support the `--auth-mode` parameter. Management operations, such as creating a resource group or storage account, automatically use Azure AD credentials for authorization. For more information, see [Choose how to authorize access to blob data with Azure CLI](#).

Run the `login` command to open a browser and connect to your Azure subscription.

```
az login
```

## Create a container

All blob data is stored within containers, so you'll need at least one container resource before you can upload data. If needed, use the following example to create a storage container. For more information, see [Managing blob containers using Azure CLI](#).

```
#!/bin/bash
storageAccount=<storage-account>
containerName="demo-container"

Create a container object
az storage container create \
--name $containerName \
--account-name $storageAccount
--auth-mode login
```

When you use the examples included in this article, you'll need to replace the placeholder values in brackets with your own values. For more information about signing into Azure with Azure CLI, see [Sign in with Azure CLI](#).

## Upload blobs

Azure CLI offers commands that perform operations on one resource or on multiple resources, depending on your requirements.

To upload a file to a block blob, pass the required parameter values to the `az storage blob upload` command. Supply the source path and file name with the `--file` parameter, and the name of the destination container with the `--container-name` parameter. You'll also need to supply the `--account-name` parameter. This command creates a new blob or overwrites the original blob if it already exists.

You can use the `az storage blob upload-batch` command to recursively upload multiple blobs to a storage container. You can use Unix filename pattern matching specify a range of files to upload with the `--pattern` parameter. The supported patterns are `*`, `?`, `[seq]`, and `[!seq]`. To learn more, refer to the Python documentation on [Unix filename pattern matching](#).

In the following example, the first operation uses the `az storage blob upload` command to upload a single, named file. The source file and destination storage container are specified with the `--file` and `--container-name` parameters.

The second operation demonstrates the use of the `az storage blob upload-batch` command to upload multiple files. The `--if-unmodified-since` parameter ensures that only files modified with the last seven days will be uploaded. The value supplied by this parameter must be provided in UTC format.

```

#!/bin/bash
storageAccount="<storage-account>"
containerName="demo-container"
lastModified=`date -d "10 days ago" '+%Y-%m-%dT%H:%MZ'`

path="C:\\temp\\"
filename="demo-file.txt"
imageFiles="*.png"
file="$path$filename"

#Upload a single named file
az storage blob upload \
--file $file \
--container-name $containerName \
--account-name $storageAccount \
--auth-mode login

#Upload multiple image files recursively
az storage blob upload-batch \
--destination $containerName \
--source $path \
--pattern *.png \
--account-name $storageAccount \
--auth-mode login \
--if-unmodified-since $lastModified

```

## List blobs

By default, the `az storage blob list` command lists all blobs stored within a container. You can use various approaches to refine the scope of your search. There's no restriction on the number of containers or blobs a storage account may have. To potentially avoid retrieving thousands of blobs, it's a good idea to limit the amount of data returned.

Use the `--prefix` parameter to select either a single known file or a range of files whose names begin with a defined string.

By default, only blobs are returned in a listing operation. In some scenarios, you may want to pass a value for the `--include` parameter to return additional types of objects such as soft-deleted blobs, snapshots, and versions. These values can be combined to return more than multiple object types.

The `--num-results` parameter can be used to limit the number of unfiltered blobs returned from a container. A service limit of 5,000 is imposed on all Azure resources. This limit ensures that manageable amounts of data are retrieved and that performance isn't impacted. If the number of blobs returned exceeds either the `--num-results` value or the service limit, a continuation token is returned. This token allows you to use multiple requests to retrieve any number of blobs. More information is available on [Enumerating blob resources](#).

The following example shows several approaches used to provide a list of blobs. The first approach lists a single blob within a specific container resource. The second approach uses the `--prefix` parameter to list all blobs in all containers with a prefix of *louis*. The search is restricted to five containers using the `--num-results` parameter. The third approach uses `--num-results` and `--marker` parameters to limit the retrieval of all blobs within a container.

For additional information, see the [az storage blob list](#) reference.

```

#!/bin/bash
storageAccount=<storage-account>
blobName="demo-file.txt"
containerName="demo-container"
blobPrefix="img-louis"
numResults=5

#Approach 1: List all blobs in a named container
az storage blob list \
 --container $containerName \
 --account-name $storageAccount \
 --prefix $blobName
 --auth-mode login

#Approach 2: Use the --prefix parameter to list blobs in all containers

containerList=$(\
 az storage container list \
 --query "[].name" \
 --num-results $numResults \
 --account-name $storageAccount \
 --auth-mode login \
 --output tsv
)
for row in $containerList
do
 tmpName=$(echo $row | sed -e 's/\r//g')
 echo $tmpName
 az storage blob list \
 --prefix $blobPrefix \
 --container $tmpName \
 --account-name $storageAccount \
 --auth-mode login
done

```

## Download a blob

Depending on your use case, you'll use either the `az storage blob download` or `az storage blob download-batch` command to download blobs. To download an individual blob, call the `az storage blob download` command directly and pass values for the `--container-name`, `--file`, and `--name` parameters. The blob will be downloaded to the shell directory by default, but an alternate location can be specified. The operation will fail with an error if your specified path doesn't exist.

To recursively download multiple blobs from a storage container, use the `az storage blob download-batch` command. This command supports Unix filename pattern matching with the `--pattern` parameter. The supported patterns are `*`, `?`, `[seq]`, and `[!seq]`. To learn more, refer to the Python documentation on [Unix filename pattern matching](#).

The following sample code provides an example of both single and multiple download approaches. It also offers a simplified approach to searching all containers for specific files using a wildcard. Because some environments may have many thousands of resources, using the `--num-results` parameter is recommended.

For additional information, see the [az storage blob download](#) and [az storage blob download batch](#) reference.

```

#!/bin/bash
#Set variables
storageAccount="<storage-account>"
containerName="demo-container"

destinationPath="C:\\temp\\downloads\\"
destinationFilename="downloadedBlob.txt"
file="$destinationPath$destinationFilename"
sourceBlobName="demo-file.txt"

#Download a single named blob

az storage blob download \
--container $containerName \
--file $file \
--name $sourceBlobName \
--account-name $storageAccount \
--auth-mode login

#Download multiple blobs using a pattern value

az storage blob download-batch \
--destination $destinationPath \
--source $containerName \
--pattern images/*.png \
--account-name $storageAccount \
--auth-mode login

#Use a loop to download matching blobs in a list of containers

containerList=$(\
 az storage container list \
 --query "[].name" \
 --num-results 5 \
 --account-name $storageAccount \
 --auth-mode login \
 --output tsv
)
for row in $containerList
do
 tmpName=$(echo $row | sed -e 's/\r//g')
 echo $tmpName

 az storage blob download-batch \
 --destination $destinationPath \
 --source $tmpName \
 --pattern *louis.* \
 --account-name $storageAccount \
 --auth-mode login
done

```

## Manage blob properties and metadata

A blob exposes both system properties and user-defined metadata. System properties exist on each Blob Storage resource. Some properties are read-only, while others can be read or set. Under the covers, some system properties map to certain standard HTTP headers.

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, and don't affect how the resource behaves.

### Reading blob properties

To read blob properties or metadata, you must first retrieve the blob from the service. Use the `az storage blob show` command to retrieve a blob's properties and metadata, but not its content. The following example retrieves a blob and lists its properties.

For additional information, see the [az storage blob show](#) reference.

```
#!/bin/bash
#Set variables
storageAccount=<storage-account>
containerName="demo-container"

az storage blob show \
 --container demo-container \
 --name demo-file.txt \
 --account-name $storageAccount \
 --auth-mode login
```

## Read and write blob metadata

Blob metadata is an optional set of name/value pairs associated with a blob. As shown in the previous example, there's no metadata associated with a blob initially, though it can be added when necessary. To read, use the `az storage blob metadata show` command. To update blob metadata, you'll use `az storage blob metadata update` and supply an array of key-value pairs. For more information, see the [az storage blob metadata](#) reference.

For additional information, see the [az storage blob metadata](#) reference.

The example below first updates and then commits a blob's metadata, and then retrieves it.

```
#!/bin/bash
storageAccount=<storage-account>
containerName="demo-container"
blobName="blue-moon.mp3"

metadata=("Written=1934" "Recorded=1958")
metadata+=("Lyricist=Lorenz Hart")
metadata+=("Composer=Richard Rogers")
metadata+=("Artist=Tony Bennett")

#Update metadata
az storage blob metadata update \
 --container-name $containerName \
 --name $blobName \
 --metadata "${metadata[@]}" \
 --account-name $storageAccount \
 --auth-mode login

#Retrieve updated blob metadata
az storage blob metadata show \
 --container-name $containerName \
 --name $blobName \
 --account-name $storageAccount \
 --auth-mode login
```

## Copy operations for blobs

There are many scenarios in which blobs of different types may be copied. Examples in this article are limited to block blobs. Azure CLI offers commands that perform operations on one resource or on multiple resources, depending on your requirements.

To copy a specific blob, use the `az storage blob copy start` command and specify values for source and destination containers and blobs. It's also possible to provide a uniform resource identifier (URI), share, or

shared access signature (SAS) as the source.

You can also specify the conditions under which the blob will be copied. These conditions can be set for either the source or destination blob. You can reference the last modified date, tag data, or ETag value. You may, for example, choose to copy blobs that haven't been recently modified to a separate container. For more information, see [Specifying conditional headers for Blob service operations](#).

You can use the `az storage blob copy start-batch` command to recursively copy multiple blobs between storage containers within the same storage account. This command requires values for the `--source-container` and `--destination-container` parameters, and can copy all files between the source and destination. Like other CLI batch commands, this command supports Unix filename pattern matching with the `--pattern` parameter. The supported patterns are `*`, `?`, `[seq]`, and `[!seq]`. To learn more, refer to the Python documentation on [Unix filename pattern matching](#).

#### NOTE

Consider the use of AzCopy for ease and performance, especially when copying blobs between storage accounts. AzCopy is a command-line utility that you can use to copy blobs or files to or from a storage account. Find out more about how to [Get started with AzCopy](#).

For more information, see the [az storage blob copy](#) reference.

The following sample code provides an example of both single and multiple copy operations. Because some environments may have many thousands of resources, using the `--num-results` parameter is recommended. The first example copies the `secret-town-road.png` blob from the `photos` container to the `locations` container. Both containers exist within the same storage account. The result verifies the success of the copy operation.

```
#!/bin/bash
storageAccount=<storage-account>
sourceContainer="photos"
blobName="secret-town-road.jpg"
destContainer="locations"

az storage blob copy start \
--destination-container $destContainer \
--destination-blob $blobName \
--source-container $sourceContainer \
--source-blob $blobName \
--account-name $storageAccount \
--auth-mode login
```

## Snapshot blobs

Any leases associated with the base blob don't affect the snapshot. You can't acquire a lease on a snapshot. Read more about [Blob snapshots](#). For more information, see the [az storage blob snapshot](#) reference.

The following sample code retrieves a blob from a storage container and creates a snapshot of it.

```
#!/bin/bash
storageAccount=<storage-account>
containerName="demo-container"
blobName="demo-file.txt"

az storage blob snapshot \
--container-name $containerName \
--name Blue-Moon.mp3 \
--account-name $storageAccount \
--auth-mode login
```

## Set blob tier

When you change a blob's tier, you move the blob and all of its data to the target tier. You can change the tier between **Hot**, **Cool**, and **Archive** with the `az storage blob set-tier` command.

Depending on your requirements, you may also utilize the *Copy Blob* operation to copy a blob from one tier to another. The *Copy Blob* operation will create a new blob in the desired tier while leaving the source blob remains in the original tier.

Changing tiers from **Cool** or **Hot** to **Archive** takes place almost immediately. After a blob is moved to the **Archive** tier, it's considered to be offline and can't be read or modified. Before you can read or modify an archived blob's data, you'll need to rehydrate it to an online tier. Read more about [Blob rehydration from the Archive tier](#).

For additional information, see the [az storage blob set-tier](#) reference.

The following sample code sets the tier to **Hot** for a single, named blob within the `archive` container.

```
#!/bin/bash
storageAccount=<storage-account>
containerName="demo-container"

az storage blob set-tier
--container-name $containerName \
--name Blue-Moon.mp3 \
--tier Hot \
--account-name $storageAccount \
--auth-mode login
```

## Operations using blob tags

Blob index tags make data management and discovery easier. Blob index tags are user-defined key-value index attributes that you can apply to your blobs. Once configured, you can categorize and find objects within an individual container or across all containers. Blob resources can be dynamically categorized by updating their index tags without requiring a change in container organization. This approach offers a flexible way to cope with changing data requirements. You can use both metadata and index tags simultaneously. For more information on index tags, see [Manage and find Azure Blob data with blob index tags](#).

### IMPORTANT

Support for blob index tags is in preview status. See the [Supplemental Terms of Use for Microsoft Azure Previews](#) for legal terms that apply to Azure features that are in beta, preview, or otherwise not yet released into general availability.

## TIP

The code sample provided below uses pattern matching to obtain text from an XML file having a known structure. The example is used to illustrate a simplified approach for adding blob tags using basic Bash functionality. The use of an actual data parsing tool is always recommended when consuming data for production workloads.

The following example illustrates how to add blob index tags to a series of blobs. The example reads data from an XML file and uses it to create index tags on several blobs. To use the sample code, create a local *blob-list.xml* file in your *C:\temp* directory. The XML data is provided below.

For additional information, see the [az storage blob set-tier](#) reference.

```
<Venue Name="House of Prime Rib" Type="Restaurant">
 <Files>
 <File path="transactions/12027121.csv" />
 <File path="campaigns/radio-campaign.docx" />
 <File path="photos/bannerphoto.png" />
 <File path="archive/completed/2020review.pdf" />
 <File path="logs/2020/01/01/logfile.txt" />
 </Files>
</Venue>
```

The sample code iterates the lines within the XML file. It locates the *Venue* element and creates variables for the *Name* and *Type* values. It then iterates through the remaining lines and creates tags for each blob referenced by a *File* node.

```
#!/bin/bash
storageAccount=<storage-account>
containerName="demo-container"

while read line
do

#Set Tag values
if echo "$line" | grep -q "<Venue";then
 name=`echo "$line" | cut -d '"' -f 2`
 type=`echo "$line" | cut -d '"' -f 4`
 tags=("name=$name")
 tags+=("type=$type")
fi

#Add tags to blobs
if echo "$line" | grep -q "<File " ;then
 blobName=`echo "$line" | cut -d '"' -f 2`

 echo az storage blob tag set \
 --container-name $containerName \
 --name $blobName \
 --account-name $storageAccount \
 --auth-mode login \
 --tags "{$tags[@]}"

fi

done < /mnt/c/temp/bloblist.xml
```

## Delete blobs

You can delete either a single blob or series of blobs with the [az storage blob delete](#) and

`az storage blob delete-batch` commands. When deleting multiple blobs, you can use conditional operations, loops, or other automation as shown in the examples below.

**WARNING**

Running the following examples may permanently delete blobs. Microsoft recommends enabling container soft delete to protect containers and blobs from accidental deletion. For more info, see [Soft delete for containers](#).

The following sample code provides an example of both individual and batch delete operations. The first example deletes a single, named blob. The second example illustrates the use of logical operations in Bash to delete multiple blobs. The third example uses the `delete-batch` command to delete all blobs with the format *bennett-x*, except *bennett-2*.

For more information, see the [az storage blob delete](#) and [az storage blob delete-batch](#) reference.

```

#!/bin/bash
storageAccount="<storage-account>"
containerName="demo-container"

blobName="demo-file.txt"
blobPrefix="sinatra-"

#Delete a single, named blob
az storage blob delete \
 --container-name $containerName \
 --name $blobName \
 --account-name $storageAccount \
 --auth-mode login

#Iterate a blob list, deleting blobs whose names end with even numbers

Get list of containers
blobList=$(az storage blob list \
 --query "[].name" \
 --prefix $blobPrefix \
 --container-name $containerName \
 --account-name $storageAccount \
 --auth-mode login \
 --output tsv)

Delete all blobs with the format *bennett-x* except *bennett-2.*
for row in $blobList
do
 #Get the blob's number
 tmpBlob=$(echo $row | sed -e 's/\r//g')
 tmpName=$(echo ${row%.*} | sed -e 's/\r//g')

 if [`expr ${tmpName}: ${#blobPrefix} % 2` == 0]
 then

 echo "Deleting $tmpBlob"
 az storage blob delete \
 --container-name $containerName \
 --name $tmpBlob \
 --account-name $storageAccount \
 --auth-mode login

 fi
done

#Delete multiple blobs using delete-batch
az storage blob delete-batch \
 --source $containerName \
 --pattern bennett-[!2].* \
 --account-name $storageAccount \
 --auth-mode login

```

In some cases, it's possible to retrieve blobs that have been deleted. If your storage account's soft delete data protection option is enabled, the `--include d` parameter and value will return blobs deleted within the account's retention period. To learn more about soft delete, refer to thee [Soft delete for blobs](#) article.

Use the following examples to retrieve a list of blobs deleted within container's associated retention period. The first example displays a list of all recently deleted blobs and the dates on which they were deleted. The second example lists all deleted blobs matching a specific prefix.

```

#!/bin/bash
storageAccount=<storage-account>
containerName="demo-container"

blobPrefix="sinatra-"

#Retrieve a list of all deleted blobs
az storage blob list \
 --container-name $containerName \
 --include d \
 --output table \
 --account-name $storageAccount \
 --auth-mode login \
 --query "[?deleted].{name:deleted:properties.deletedTime}"

#Retrieve a list of all deleted blobs matching a specific prefix
az storage blob list \
 --container-name $containerName \
 --prefix $blobPrefix \
 --output table \
 --include d \
 --account-name $storageAccount \
 --auth-mode login \
 --query "[].{name:deleted}"

```

## Restore a deleted blob

As mentioned in the [List blobs](#) section, you can configure the soft delete data protection option on your storage account. When enabled, it's possible to restore containers deleted within the associated retention period. You may also use versioning to maintain previous versions of your blobs for each recovery and restoration.

If blob versioning and blob soft delete are both enabled, then modifying, overwriting, deleting, or restoring a blob automatically creates a new version. The method you'll use to restore a deleted blob will depend upon whether versioning is enabled on your storage account.

The following code sample restores all soft-deleted blobs or, if versioning is enabled, restores the latest version of a blob. It first determines whether versioning is enabled with the

```
az storage account blob-service-properties show
```

If versioning is enabled, the `az storage blob list` command retrieves a list of all uniquely-named blob versions. Next, the blob versions on the list are retrieved and ordered by date. If no versions are found with the `isCurrentVersion` attribute value, the `az storage blob copy start` command is used to make an active copy of the blob's latest version.

If versioning is disabled, the `az storage blob undelete` command is used to restore each soft-deleted blob in the container.

Before you can follow this example, you'll need to enable soft delete on at least one of your storage accounts. To learn more about the soft delete data protection option, refer to the [Soft delete for blobs](#) article or the [az storage blob undelete](#) reference.

```

#!/bin/bash
storageAccount=<storage-account>
groupName="myResourceGroup"
containerName="demo-container"

blobSvcProps=$(
 az storage account blob-service-properties show \
 --account-name $storageAccount \
 --resource-group $groupName)

```

```

softDelete=$(echo "${blobSvcProps}" | jq -r '.deleteRetentionPolicy.enabled')
versioning=$(echo "${blobSvcProps}" | jq -r '.isVersioningEnabled')

If soft delete is enabled
if $softDelete
then

 # If versioning is enabled
 if $versioning
 then

 # Get all blobs and versions using -Unique to avoid processing duplicates/versions
 blobList=$(
 az storage blob list \
 --account-name $storageAccount \
 --container-name $containerName \
 --include dv --query "[?versionId != null].{name:name}" \
 --auth-mode login -o tsv | uniq)

 # Iterate the collection
 for blob in $blobList
 do
 # Get all versions of the blob, newest to oldest
 blobVers=$(
 az storage blob list \
 --account-name $storageAccount \
 --container-name $containerName \
 --include dv \
 --prefix $blob \
 --auth-mode login -o json | jq 'sort_by(.versionId) | reverse | .[]')
 # Select the first (newest) object
 delBlob=$(echo "$blobVers" | jq -sr '[.[]][0]')

 # Verify that the newest version is NOT the latest (that the version is "deleted")
 if [[$(echo "$delBlob" | jq '.isCurrentVersion') != true]]; then
 # Get the blob's versionId property, build the URI to the blob
 versionID=$(echo "$delBlob" | jq -r '.versionId')
 uri="https://$storageAccount.blob.core.windows.net/$containerName/$blob?
versionId=$versionID"

 # Copy the latest version
 az storage blob copy start \
 --account-name $storageAccount \
 --destination-blob $blob \
 --destination-container $containerName \
 --source-uri $uri \
 --auth-mode login

 delBlob=""
 fi
 done
 else
 #Retrieve all deleted blobs
 blobList=$(\
 az storage blob list \
 --container-name $containerName \
 --include d \
 --output tsv \
 --account-name $storageAccount \
 --auth-mode login \
 --query "[?deleted].[name]" \
)

 #Iterate list of deleted blobs and restore
 for row in $blobList
 do

```

```
 tmpName=$(echo $row | sed -e 's/\r//g')
 echo "Restoring $tmpName"
 az storage blob undelete \
 --container-name $containerName \
 --name $tmpName \
 --account-name $storageAccount \
 --auth-mode login
 done

fi

else

#Soft delete is not enabled
echo "Sorry, the delete retention policy is not enabled."

fi
```

## Next steps

- [Choose how to authorize access to blob data with Azure CLI](#)
- [Run PowerShell commands with Azure AD credentials to access blob data](#)
- [Manage blob containers using CLI](#)

# Manage block blobs with PowerShell

8/22/2022 • 19 minutes to read • [Edit Online](#)

Blob storage supports block blobs, append blobs, and page blobs. Block blobs are optimized for uploading large amounts of data efficiently. Block blobs are ideal for storing images, documents, and other types of data that isn't subjected to random read and write operations. This article explains how to work with block blobs.

## Prerequisites

- An Azure subscription. See [Get Azure free trial](#).
- Azure PowerShell module Az, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#).

### Configure a context object to encapsulate credentials

Every request to Azure Storage must be authorized. You can authorize a request made from PS with your Azure AD account or by using the account access keys. The examples in this article use Azure AD authorization in conjunction with context objects. Context objects encapsulate your Azure AD credentials and pass them during subsequent data operations.

To sign in to your Azure account with an Azure AD account, open PowerShell and call the [Connect-AzAccount](#) cmdlet.

```
#Connect to your Azure subscription
Connect-AzAccount
```

After the connection has been established, create the Azure context. Authenticating with Azure AD automatically creates an Azure context for your default subscription. In some cases, you may need to access resources in a different subscription after authenticating. To accomplish this, you can change the subscription associated with your current Azure session by modifying the active session context.

To use your default subscription, create the context by calling the `New-AzStorageContext` cmdlet. Include the `-UseConnectedAccount` parameter so that data operations will be performed using your Azure AD credentials.

```
#Create a context object using Azure AD credentials
$ctx = New-AzStorageContext -StorageAccountName <storage account name> -UseConnectedAccount
```

To change subscriptions, retrieve the context object with the [Get-AzSubscription](#) cmdlet, then change the current context with the [Set-AzContext](#). For more information, see [Change the active subscription](#).

### Create a container

All blob data is stored within containers, so you'll need at least one container resource before you can upload data. If needed, use the following example to create a storage container. For more information, see [Managing blob containers using PowerShell](#).

```
#Create a container object
$container = New-AzStorageContainer -Name "myContainer" -Context $ctx
```

When you use the following examples, you'll need to replace the placeholder values in brackets with your own values. For more information about signing into Azure with PowerShell, see [Sign in with Azure PowerShell](#).

## Upload a blob

To upload a file to a block blob, pass the required parameter values to the `Set-AzStorageBlobContent` cmdlet.

Supply the path and file name with the `-File` parameter, and the name of the container with the `-Container` parameter. You'll also need to provide a reference to the context object with the `-Context` parameter.

This command creates the blob if it doesn't exist, or prompts for overwrite confirmation if it exists. You can overwrite the file without confirmation if you pass the `-Force` parameter to the cmdlet.

The following example specifies a `-File` parameter value to upload a single, named file. It also demonstrates the use of the PowerShell pipeline operator and `Get-ChildItem` cmdlet to upload multiple files. The `Get-ChildItem` cmdlet uses the `-Path` parameter to specify `C:\Temp\*.png`. The inclusion of the asterisk (`*`) wildcard specifies all files with the `.png` filename extension. The `-Recurse` parameter searches the `Temp` directory and its subdirectories.

```
#Set variables
$path = "C:\temp\"
$containerName = "myContainer"
$filename = "demo-file.txt"
$imageFiles = $path + "*.png"
$file = $path + $filename

#Upload a single named file
Set-AzStorageBlobContent -File $file -Container $containerName -Context $ctx

#Upload multiple image files recursively
Get-ChildItem -Path $imageFiles -Recurse | Set-AzStorageBlobContent -Container $containerName -Context $ctx
```

The result displays the storage account name, the storage container name, and provides a list of the files uploaded.

```
AccountName: demostorageaccount, ContainerName: demo-container
```

Name	BlobType	Length	ContentType	LastModified	AccessTier	IsDeleted
---	-----	-----	-----	-----	-----	-----
demo-file.txt	BlockBlob	222	application/octet-stream	2021-12-14 01:38:03Z	Cool	False
hello-world.png	BlockBlob	14709	application/octet-stream	2021-12-14 01:38:03Z	Cool	False
hello-world2.png	BlockBlob	12472	application/octet-stream	2021-12-14 01:38:03Z	Cool	False
hello-world3.png	BlockBlob	13537	application/octet-stream	2021-12-14 01:38:03Z	Cool	False

## List blobs

The `Get-AzStorageBlob` cmdlet is used to list blobs stored within a container. You can use various approaches to define the scope of your search. Use the `-Container` and `-Name` parameter to list a specific blob within a known container. To generate an unfiltered list of all blobs within a specific container, use the `-Container` parameter alone, without a `-Name` value.

There's no restriction on the number of containers or blobs a storage account may have. To potentially avoid retrieving thousands of blobs, it's a good idea to limit the amount of data returned. When retrieving multiple blobs, you can use the `-Prefix` parameter to specify blobs whose names begin with a specific string. You may also use the `-Name` parameter in conjunction with a wildcard to specify file names or types.

The `-MaxCount` parameter can be used to limit the number of unfiltered blobs returned from a container. A service limit of 5,000 is imposed on all Azure resources. This limit ensures that manageable amounts of data are retrieved and performance is not impacted. If the number of blobs returned exceeds either the `-MaxCount` value or the service limit, a continuation token is returned. This token allows you to use multiple requests to retrieve

any number of blobs. More information is available on [Enumerating blob resources](#).

The following example shows several approaches used to provide a list of blobs. The first approach lists a single blob within a specific container resource. The second approach uses a wildcard to list all `.jpg` files with a prefix of `/louis`. The search is restricted to five containers using the `-MaxCount` parameter. The third approach uses `-MaxCount` and `-ContinuationToken` parameters to limit the retrieval of all blobs within a container.

```
#Set variables
$namedContainer = "named-container"
$demoContainer = "myContainer"
$containerPrefix = "demo"

$maxCount = 1000
$total = 0
$token = $Null

#Approach 1: List all blobs in a named container
Get-AzStorageBlob -Container $namedContainer -Context $ctx

#Approach 2: Use a wildcard to list blobs in all containers
Get-AzStorageContainer -MaxCount 5 -Context $ctx | Get-AzStorageBlob -Blob "*louis*.jpg"

#Approach 3: List batches of blobs using MaxCount and ContinuationToken parameters
Do
{
 #Retrieve blobs using the MaxCount parameter
 $blobs = Get-AzStorageBlob -Container $demoContainer -MaxCount $maxCount -ContinuationToken $token -Context $ctx
 $blobCount = 1

 #Loop through the batch
 Foreach ($blob in $blobs)
 {
 #To-do: Perform some work on individual blobs here

 #Display progress bar
 $percent = $($blobCount/$maxCount*100)
 Write-Progress -Activity "Processing blobs" -Status "$percent% Complete" -PercentComplete $percent
 $blobCount++
 }

 #Update $total
 $total += $blobs.Count

 #Exit if all blobs processed
 If($blobs.Length -le 0) { Break; }

 #Set continuation token to retrieve the next batch
 $token = $blobs[$blobs.Count -1].ContinuationToken
}
While ($null -ne $token)
Write-Host "`n`n AccountName: $($ctx.StorageAccountName), ContainerName: $demoContainer `n"
Write-Host "Processed $total blobs in $namedContainer."
```

The first two approaches display the storage account and container names, and a list of the blobs retrieved. The third approach displays the total count of blobs within a named container. The blobs are retrieved in batches, and a status bar shows the progress during the count.

```
AccountName: demostorageaccount, ContainerName: named-container
```

Name	BlobType	Length	ContentType	LastModified	AccessTier	IsDeleted
index.txt	BlockBlob	222	text/plain	2021-12-15 22:00:10Z	Cool	False
miles-davis.txt	BlockBlob	23454	text/plain	2021-12-15 22:17:59Z	Cool	False
cab-calloway.txt	BlockBlob	18419	text/plain	2021-12-15 22:17:59Z	Cool	False
benny-goodman.txt	BlockBlob	17726	text/plain	2021-12-15 22:17:59Z	Cool	False

```
AccountName: demostorageaccount, ContainerName: demo-container
```

Name	BlobType	Length	ContentType	LastModified	AccessTier	IsDeleted
louis-armstrong.jpg	BlockBlob	211482	image/jpeg	2021-12-14 01:38:03Z	Cool	False
louis-jordan.jpg	BlockBlob	55766	image/jpeg	2021-12-14 01:38:03Z	Cool	False
louis-prima.jpg	BlockBlob	290651	image/jpeg	2021-12-14 01:38:03Z	Cool	False

```
AccountName: demostorageaccount, ContainerName: demo-container
```

```
Processed 5257 blobs in demo-container.
```

## Download a blob

Depending on your use case, the `Get-AzStorageBlobContent` cmdlet can be used to download either single or multiple blobs. As with most operations, both approaches require a context object.

To download a single named blob, you can call the cmdlet directly and supply values for the `-Blob` and `-Container` parameters. The blob will be downloaded to the working PowerShell directory by default, but an alternate location can be specified. To change the target location, a valid, existing path must be passed with the `-Destination` parameter. Because the operation can't create a destination, it will fail with an error if your specified path doesn't exist.

Multiple blobs can be downloaded by combining the `Get-AzStorageBlob` cmdlet and the PowerShell pipeline operator. First, create a list of blobs with the `Get-AzStorageBlob` cmdlet. Next, use the pipeline operator and the `Get-AzStorageBlobContent` cmdlet to retrieve the blobs from the container.

The following sample code provides an example of both single and multiple download approaches. It also offers a simplified approach to searching all containers for specific files using a wildcard. Because some environments may have hundreds of thousands of resources, using the `-MaxCount` parameter is recommended.

```
#Set variables
$containerName = "myContainer"
$path = "C:\temp\downloads\
$blobName = "demo-file.txt"
$fileList = "*.png"
$pipelineList = "louis*"
$maxCount = 10

#Download a single named blob
Get-AzStorageBlobContent -Container $containerName -Blob $blobName -Destination $path -Context $ctx

#Download multiple blobs using the pipeline
Get-AzStorageBlob -Container $containerName -Blob $fileList -Context $ctx | Get-AzStorageBlobContent

#Use wildcard to download blobs from all containers
Get-AzStorageContainer -MaxCount $maxCount -Context $ctx | Get-AzStorageBlob -Blob "louis*" | Get-AzStorageBlobContent
```

The result displays the storage account and container names and provides a list of the files downloaded.

AccountName: demostorageaccount, ContainerName: demo-container						
Name IsDeleted	BlobType	Length	ContentType	LastModified	AccessTier	
--	-----	-----	-----	-----	-----	-----
demo-file.txt	BlockBlob	222	application/octet-stream	2021-12-14 01:38:03Z	Unknown	False
hello-world.png	BlockBlob	14709	application/octet-stream	2021-12-14 01:38:03Z	Unknown	False
hello-world2.png	BlockBlob	12472	application/octet-stream	2021-12-14 01:38:03Z	Unknown	False
hello-world3.png	BlockBlob	13537	application/octet-stream	2021-12-14 01:38:03Z	Unknown	False
AccountName: demostorageaccount, ContainerName: public-container						
Name IsDeleted	BlobType	Length	ContentType	LastModified	AccessTier	
--	-----	-----	-----	-----	-----	-----
louis-armstrong.jpg	BlockBlob	211482	image/jpeg	2021-12-14 18:56:03Z	Unknown	False
AccountName: demostorageaccount, ContainerName: read-only-container						
Name IsDeleted	BlobType	Length	ContentType	LastModified	AccessTier	
--	-----	-----	-----	-----	-----	-----
louis-jordan.jpg	BlockBlob	55766	image/jpeg	2021-12-14 18:56:21Z	Unknown	False
AccountName: demostorageaccount, ContainerName: hidden-container						
Name IsDeleted	BlobType	Length	ContentType	LastModified	AccessTier	
--	-----	-----	-----	-----	-----	-----
louis-prima.jpg	BlockBlob	290651	image/jpeg	2021-12-14 18:56:45Z	Unknown	False

## Manage blob properties and metadata

A container exposes both system properties and user-defined metadata. System properties exist on each Blob Storage resource. Some properties are read-only, while others can be read or set. Under the covers, some system properties map to certain standard HTTP headers.

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.

### Reading blob properties

To read blob properties or metadata, you must first retrieve the blob from the service. Use the

`Get-AzStorageBlob` cmdlet to retrieve a blob's properties and metadata, but not its content. Next, use the `BlobClient.GetProperties()` method to fetch the blob's properties. The properties or metadata can then be read or set as needed.

The following example retrieves a blob and lists its properties.

```
$blob = Get-AzStorageBlob -Blob "blue-moon.mp3" -Container "myContainer" -Context $ctx
$properties = $blob.BlobClient.GetProperties()
Echo $properties.Value
```

The result displays a list of the blob's properties as shown below.

```

LastModified : 11/16/2021 3:42:07 PM +00:00
CreatedOn : 11/16/2021 3:42:07 PM +00:00
Metadata : {}
BlobType : Block
LeaseDuration : Infinite
LeaseState : Available
LeaseStatus : Unlocked
ContentLength : 2163298
ContentType : audio/mpeg
ETag : 0x8D9C0AA9E0CBA78
IsServerEncrypted : True
AccessTier : Cool
IsLatestVersion : False
TagCount : 0
ExpiresOn : 1/1/0001 12:00:00 AM +00:00
LastAccessed : 1/1/0001 12:00:00 AM +00:00
HasLegalHold : False

```

## Read and write blob metadata

Blob metadata is an optional set of name/value pairs associated with a blob. As shown in the previous example, there's no metadata associated with a blob initially, though it can be added when necessary. To update blob metadata, you'll use the `BlobClient.UpdateMetadata` method. This method only accepts key-value pairs stored in a generic `IDictionary` object. For more information, see the [BlobClient](#) class definition.

The example below first updates and then commits a blob's metadata, and then retrieves it. The sample blob is flushed from memory to ensure the metadata isn't being read from the in-memory object.

```

#Set variable
$container = "myContainer"
$blobName = "blue-moon.mp3"

#Retrieve blob
blob = Get-AzStorageBlob -Blob $blobName -Container $container -Context $ctx

#Create IDictionary, add key-value metadata pairs to IDictionary
$metadata = New-Object System.Collections.Generic.Dictionary[String, String]
$metadata.Add("YearWritten", "1934")
$metadata.Add("YearRecorded", "1958")
$metadata.Add("Composer", "Richard Rogers")
$metadata.Add("Lyricist", "Lorenz Hart")
$metadata.Add("Artist", "Tony Bennett")

#Update metadata
blob.BlobClient.SetMetadata($metadata, $null)

#Flush blob from memory, retrieve updated blob, retrieve properties
$blob = $null
blob = Get-AzStorageBlob -Blob $blobName -Container $container -Context $ctx
$properties = $blob.BlobClient.GetProperties()

#Display metadata
Echo $properties.Value.Metadata

```

The result returns the blob's newly updated metadata as shown below.

Key	Value
---	-----
YearWritten	1934
YearRecorded	1958
Composer	Richard Rogers
Lyricist	Lorenz Hart
Artist	Tony Bennett

## Copy operations for blobs

There are many scenarios in which blobs of different types may be copied. Examples in this article are limited to block blobs.

### Copy a source blob to a destination blob

For a simplified copy operation within the same storage account, use the `Copy-AzStorageBlob` cmdlet. Because the operation is copying a blob within the same storage account, it's a synchronous operation. Cross-account operations are asynchronous.

You should consider the use of AzCopy for ease and performance, especially when copying blobs between storage accounts. AzCopy is a command-line utility that you can use to copy blobs or files to or from a storage account. Find out more about how to [Get started with AzCopy](#).

The example below copies the `bannerphoto.png` blob from the `photos` container to the `photos` folder within the `archive` container. Both containers exist within the same storage account. The result verifies the success of the copy operation.

```
$blobname = "bannerphoto.png"
Copy-AzStorageBlob -SrcContainer "photos" -SrcBlob $blobname -DestContainer "archive" -DestBlob
$("photos/$blobname") -Context $ctx

AccountName: demostorageaccount, ContainerName: archive

Name BlobType Length ContentType LastModified AccessTier SnapshotTime
IsDeleted VersionId
---- ----- ----- ----- ----- ----- -----
- -
photos/bannerphoto BlockBlob 12472 image/png 2021-11-27 23:11:43Z Cool False
```

You can use the `-Force` parameter to overwrite an existing blob with the same name at the destination. This operation effectively replaces the destination blob. It also removes any uncommitted blocks and overwrites the destination blob's metadata.

### Copy a snapshot to a destination blob with a different name

The resulting destination blob is a writeable blob and not a snapshot.

The source blob for a copy operation may be a block blob, an append blob, a page blob, or a snapshot. If the destination blob already exists, it must be of the same blob type as the source blob. An existing destination blob will be overwritten.

The destination blob can't be modified while a copy operation is in progress. A destination blob can only have one outstanding copy operation. In other words, a blob can't be the destination for multiple pending copy operations.

When you copy a blob within the same storage account, it's a synchronous operation. When you copy across accounts it's an asynchronous operation.

The entire source blob or file is always copied. Copying a range of bytes or set of blocks is not supported.

When a blob is copied, its system properties are copied to the destination blob with the same values.

It also shows how to abort an asynchronous copy operation.

## Snapshot blobs

A snapshot is a read-only version of a blob that's taken at a point in time. A snapshot of a blob is identical to its base blob, except that the blob URI has a DateTime value appended to the blob URI to indicate the time at which the snapshot was taken. The only distinction between the base blob and the snapshot is the appended DateTime value.

Any leases associated with the base blob do not affect the snapshot. You cannot acquire a lease on a snapshot. Read more about [Blob snapshots](#).

The following sample code retrieves a blob from a storage container and creates a snapshot of it.

```
$blob = Get-AzStorageBlob -Container "manuscripts" -Blob "novels/fast-cars.docx" -Context $ctx
blob.BlobClient.CreateSnapshot()
```

## Set blob tier

When you change a blob's tier, you move the blob and all of its data to the target tier. To do this, retrieve a blob with the `Get-AzStorageBlob` cmdlet, and call the `BlobClient.SetAccessTier` method. This can be used to change the tier between **Hot**, **Cool**, and **Archive**.

Changing tiers from **Cool** or **Hot** to **Archive** take place almost immediately. After a blob is moved to the **Archive** tier, it's considered to be offline and can't be read or modified. Before you can read or modify an archived blob's data, you'll need to rehydrate it to an online tier. Read more about [Blob rehydration from the Archive tier](#).

The following sample code sets the tier to **Hot** for all blobs within the `archive` container.

```
$blobs = Get-AzStorageBlob -Container archive -Context $ctx
Foreach($blob in $blobs) {
 $blob.BlobClient.SetAccessTier("Hot")
}
```

## Operations using blob tags

Blob index tags makes data management and discovery easier. Blob index tags are user-defined key-value index attributes that you can apply to your blobs. Once configured, you can categorize and find objects within an individual container or across all containers. Blob resources can be dynamically categorized by updating their index tags without requiring a change in container organization. This offers a flexible way to cope with changing data requirements. You can use both metadata and index tags simultaneously. For more information on index tags, see [Manage and find Azure Blob data with blob index tags](#).

The following example illustrates how to add blob index tags to a series of blobs. The example reads data from an XML file and uses it to create index tags on several blobs. To use the sample code, create a local `blob-list.xml` file in your `C:\temp` directory. The XML data is provided below.

```

<Venue Name="House of Prime Rib" Type="Restaurant">
 <Files>
 <File path="transactions/12027121.csv" />
 <File path="campaigns/radio-campaign.docx" />
 <File path="photos/bannerphoto.png" />
 <File path="archive/completed/2020review.pdf" />
 <File path="logs/2020/01/01/logfile.txt" />
 </Files>
</Venue>

```

The sample code creates a hash table and assigns the `$tags` variable to it. Next, it uses the `Get-Content` and `Get-Data` cmdlets to create an object based on the XML structure. It then adds key-value pairs to the hash table to be used as the tag values. Finally, it iterates through the XML object and creates tags for each `File` node.

```

#Set variables
$filePath = "C:\temp\blob-list.xml"
/tags = @{}

#Get data, set tag key-values
[xml]$data = Get-Content -Path $filepath
$tags.Add("VenueName", $data.Venue.Name)
$tags.Add("VenueType", $data.Venue.Type)

#Loop through files and add tag
$data.Venue.Files.ChildNodes | ForEach-Object {
 #break the path: container name, blob
 $path = $_.Path -split "/",2

 #set apply the blob tags
 Set-AzStorageBlobTag -Container $location[0] -Blob $location[1] -Tag $tags -Context $ctx
}

```

## Delete blobs

You can delete either a single blob or series of blobs with the `Remove-AzStorageBlob` cmdlet. When deleting multiple blobs, you can leverage conditional operations, loops, or the PowerShell pipeline as shown in the examples below.

### WARNING

Running the following examples may permanently delete blobs. Microsoft recommends enabling container soft delete to protect containers and blobs from accidental deletion. For more info, see [Soft delete for containers](#).

```

#Create variables
$containerName = "myContainer"
$blobName = "demo-file.txt"
$prefixName = "file"

#Delete a single, named blob
Remove-AzStorageBlob -Blob $blobName -Container $containerName -Context $ctx

#Iterate a loop, deleting blobs
for ($i = 1; $i -le 3; $i++) {
 Remove-AzStorageBlob -Blob (-join($prefixName, $i, ".txt")) -Container $containerName -Context $ctx
}

#Retrieve blob list, delete using a pipeline
Get-AzStorageBlob -Prefix $prefixName -Container $containerName -Context $ctx | Remove-AzStorageBlob

```

In some cases, it's possible to retrieve blobs that have been deleted. If your storage account's soft delete data protection option is enabled, the `-IncludeDeleted` parameter will return blobs deleted within the associated retention period. To learn more about soft delete, refer to the [Soft delete for blobs](#) article.

Use the following example to retrieve a list of blobs deleted within container's associated retention period. The result displays a list of recently deleted blobs.

```
#Retrieve a list of blobs including those recently deleted
Get-AzStorageBlob -Prefix $prefixName -IncludeDeleted -Context $ctx

AccountName: demostorageaccount, ContainerName: demo-container

Name BlobType Length ContentType LastModified AccessTier IsDeleted
---- ----- ----- ----- ----- ----- -----
file.txt BlockBlob 22 application/octet-stream 2021-12-16 20:59:41Z Cool True
file2.txt BlockBlob 22 application/octet-stream 2021-12-17 00:14:24Z Cool True
file3.txt BlockBlob 22 application/octet-stream 2021-12-17 00:14:24Z Cool True
file4.txt BlockBlob 22 application/octet-stream 2021-12-17 00:14:25Z Cool True
```

## Restore a deleted blob

As mentioned in the [List blobs](#) section, you can configure the soft delete data protection option on your storage account. When enabled, it's possible to restore blobs deleted within the associated retention period. You may also use versioning to maintain previous versions of your blobs for each recovery and restoration.

If blob versioning and blob soft delete are both enabled, then modifying, overwriting, deleting, or restoring a blob automatically creates a new version. The method you'll use to restore a deleted blob will depend upon whether versioning is enabled on your storage account.

The following code sample restores all soft-deleted blobs or, if versioning is enabled, restores the latest version of a blob. It first determines whether versioning is enabled with the `Get-AzStorageBlobServiceProperty` cmdlet.

If versioning is enabled, the `Get-AzStorageBlob` cmdlet retrieves a list of all uniquely-named blob versions. Next, the blob versions on the list are retrieved and ordered by date. If no versions are found with the `LatestVersion` attribute value, the `Copy-AzBlob` cmdlet is used to make an active copy of the latest version.

If versioning is disabled, the `BlobBaseClient.Undelete` method is used to restore each soft-deleted blob in the container.

Before you can follow this example, you'll need to enable soft delete or versioning on at least one of your storage accounts.

To learn more about the soft delete data protection option, refer to the [Soft delete for blobs](#) article.

```
$accountName = "myStorageAccount"
$groupName = "myResourceGroup"
$containerName = "myContainer"

$blobSvc = Get-AzStorageBlobServiceProperty `
 -StorageAccountName $accountName `
 -ResourceGroupName $groupName

If soft delete is enabled
if($blobSvc.DeleteRetentionPolicy.Enabled)
{
 # If versioning is enabled
 if($blobSvc.IsVersioningEnabled -eq $true)
 {
 # Set context
 $ctx = New-AzStorageContext -StorageAccountName $accountName -ResourceGroupName $groupName
```

```

$ctx = New-AzStorageContext
 -StorageAccountName $accountName `
 -UseConnectedAccount

Get all blobs and versions using -Unique
to avoid processing duplicates/versions
$blobs = Get-AzStorageBlob `
 -Container $containerName `
 -Context $ctx -IncludeVersion | `
 Where-Object {$_.VersionId -ne $null} | `
 Sort-Object -Property Name -Unique

Iterate the collection
foreach ($blob in $blobs)
{
 # Process versions
 if($blob.VersionId -ne $null)
 {

 # Get all versions of the blob, newest to oldest
 $delBlob = Get-AzStorageBlob `
 -Container $containerName `
 -Context $ctx `
 -Prefix $blob.Name `
 -IncludeDeleted -IncludeVersion | `
 Sort-Object -Property VersionId -Descending

 # Verify that the newest version is NOT the latest (that the version is "deleted")
 if (-Not $delBlob[0].IsLatestVersion)
 {
 $delBlob[0] | Copy-AzStorageBlob `
 -DestContainer $containerName `
 -DestBlob $delBlob[0].Name
 }
 }

 #Dispose the temporary object
 $delBlob = $null
}

}

}

Otherwise (if versioning is disabled)
else
{
 $blobs = Get-AzStorageBlob `
 -Container $containerName `
 -Context $ctx -IncludeDeleted | `
 Where-Object {$_.IsDeleted}
 foreach($blob in $blobs)
 {
 if($blob.IsDeleted) { $blob.BlobBaseClient.Undelete() }
 }
}
else
{
 echo "Sorry, the delete retention policy is not enabled."
}

```

## Next steps

- Run PowerShell commands with Azure AD credentials to access blob data

- [Create a storage account](#)
- [Manage blob containers using PowerShell](#)

# Use blob index tags to manage and find data on Azure Blob Storage

8/22/2022 • 6 minutes to read • [Edit Online](#)

Blob index tags categorize data in your storage account using key-value tag attributes. These tags are automatically indexed and exposed as a searchable multi-dimensional index to easily find data. This article shows you how to set, get, and find data using blob index tags.

To learn more about this feature along with known issues and limitations, see [Manage and find Azure Blob data with blob index tags](#).

## Upload a new blob with index tags

This task can be performed by a [Storage Blob Data Owner](#) or a security principal that has been given permission to the `Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write` [Azure resource provider operation](#) via a custom Azure role.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [AzCopy](#)

1. In the [Azure portal](#), select your storage account.
2. Navigate to the **Containers** option under **Data storage**, and select your container.
3. Select the **Upload** button and browse your local file system to find a file to upload as a block blob.
4. Expand the **Advanced** dropdown and go to the **Blob Index Tags** section.
5. Input the key/value blob index tags that you want applied to your data.
6. Select the **Upload** button to upload the blob.

The screenshot shows the Azure Storage Blob upload interface. On the left, there's a list of blobs with a single entry: "blob-index-sample/blob.txt". On the right, the "Upload blob" form is displayed. In the "Files" section, "blob.txt" is selected. Below it, the "Advanced" settings are shown, including "Authentication type" (set to "Account key"), "Blob type" (set to "Block blob"), and "Access tier" (set to "Hot (inferred)"). A red box highlights the "Blob Index Tags" section, which contains a table with three rows: "Status" (Value: "Unprocessed"), "Priority" (Value: "01"), and "Project" (Value: "Contoso"). The "Upload" button is at the bottom.

## Get, set, and update blob index tags

Getting blob index tags can be performed by a [Storage Blob Data Owner](#) or a security principal that has been given permission to the `Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/read` [Azure resource provider operation](#) via a custom Azure role.

Setting and updating blob index tags can be performed by a [Storage Blob Data Owner](#) or a security principal that has been given permission to the

`Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write` [Azure resource provider operation](#) via a custom Azure role.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [AzCopy](#)

1. In the [Azure portal](#), select your storage account.
2. Navigate to the **Containers** option under **Data storage**, select your container.
3. Select your blob from the list of blobs within the selected container.
4. The blob overview tab will display your blob's properties including any **Blob Index Tags**.
5. You can get, set, modify, or delete any of the key/value index tags for your blob.
6. Select the **Save** button to confirm any updates to your blob.

The screenshot shows the Azure Storage Blob properties page for a blob named 'blob.txt'. The 'Properties' section displays various metadata such as URL, Last Modified, Creation Time, Type (Block blob), and Access Tier (Hot (Inferred)). Below the properties is a blue 'Undelete' button. The 'Metadata' section is empty. The 'Blob Index Tags' section is highlighted with a red box and contains the following table:

Key	Value
Priority	01
Project	Contoso
Status	Processed

## Filter and find data with blob index tags

This task can be performed by a [Storage Blob Data Owner](#) or a security principal that has been given permission to the `Microsoft.Storage/storageAccounts/blobServices/containers/blobs/filter/action` [Azure resource provider operation](#) via a custom Azure role.

### NOTE

You can't use index tags to retrieve previous versions. Tags for previous versions aren't passed to the blob index engine. For more information, see [Conditions and known issues](#).

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [AzCopy](#)

Within the Azure portal, the blob index tags filter automatically applies the `@container` parameter to scope your selected container. If you wish to filter and find tagged data across your entire storage account, use our REST API, SDKs, or tools.

1. In the [Azure portal](#), select your storage account.
2. Navigate to the **Containers** option under **Data storage**, select your container.

3. Select the **Blob Index tags filter** button to filter within the selected container.

4. Enter a blob index tag key and tag value.

5. Select the **Blob Index tags filter** button to add additional tag filters (up to 10).

The screenshot shows the Azure Storage Explorer interface for a container named "blob-index-sample". The left sidebar has sections for Overview, Access Control (IAM), Settings (Access policy, Properties, Metadata, Editor (preview)), and a search bar. The main area shows a list of blobs with their names: blob.txt, BlobIndex.xlsx, BlobSamples.docx, logFiles.log, virtualFolder/blobImage.png, virtualFolder/Blobs.bmp, and virtualFolder/storage.jpg. Above the blob list, there are several filter buttons: Project == Contoso, Status == Unprocessed, Source == ClientUploaded, and a button labeled "Blob Index tags filter" which is highlighted with a red box. The status bar at the bottom indicates "Authentication method: Access key (Switch to Azure AD User Account)" and "Location: blob-index-sample".

## Next steps

- Learn more about blob index tags, see [Manage and find Azure Blob data with blob index tags](#)
- Learn more about lifecycle management, see [Manage the Azure Blob Storage lifecycle](#)

# Enable Azure Storage blob inventory reports

8/22/2022 • 3 minutes to read • [Edit Online](#)

The Azure Storage blob inventory feature provides an overview of your containers, blobs, snapshots, and blob versions within a storage account. Use the inventory report to understand various attributes of blobs and containers such as your total data size, age, encryption status, immutability policy, and legal hold and so on. The report provides an overview of your data for business and compliance requirements.

To learn more about blob inventory reports, see [Azure Storage blob inventory](#).

Enable blob inventory reports by adding a policy with one or more rules to your storage account. Add, edit, or remove a policy by using the [Azure portal](#).

## Enable inventory reports

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

1. Sign in to the [Azure portal](#) to get started.
2. Locate your storage account and display the account overview.
3. Under **Data management**, select **Blob inventory**.
4. Select **Add your first inventory rule**.

The **Add a rule** page appears.

5. In the **Add a rule** page, name your new rule.
6. Choose a container.
7. Under **Object type to inventory**, choose whether to create a report for blobs or containers.

If you select **Blob**, then under **Blob subtype**, choose the types of blobs that you want to include in your report, and whether to include blob versions and/or snapshots in your inventory report.

### NOTE

Versions and snapshots must be enabled on the account to save a new rule with the corresponding option enabled.

8. Select the fields that you would like to include in your report, and the format of your reports.
9. Choose how often you want to generate reports.
10. Optionally, add a prefix match to filter blobs in your inventory report.
11. Select **Save**.

The screenshot shows the Microsoft Azure portal interface for a storage account named 'contoso'. On the left, there's a navigation sidebar with sections like 'Data management', 'Settings', and 'Monitoring'. The 'Blob inventory (preview)' option is highlighted with a red box. The main content area displays a 'See an' card with the text 'Add your first rule in order to ...'. To the right, a modal window titled 'Add a rule' is open, prompting for configuration details. The fields include:

- Rule name \***: myrule
- Container \***: mycontainer
- Object type to inventory**: Blob (selected)
- Blob types**: Block blobs, Page blobs, Append blobs (unchecked)
- Blob subtype**: Include blob versions, Include snapshots (unchecked)
- Blob inventory fields \***: 3 selected
- Inventory frequency**: Daily (selected)
- Export format**: CSV (selected)
- Prefix match**: Enter a prefix (text input field)

At the bottom of the modal are 'Save' and 'Cancel' buttons.

## Next steps

- Calculate the count and total size of blobs per container
- Manage the Azure Blob Storage lifecycle

# Calculate blob count and total size per container using Azure Storage inventory

8/22/2022 • 2 minutes to read • [Edit Online](#)

This article uses the Azure Blob Storage inventory feature and Azure Synapse to calculate the blob count and total size of blobs per container. These values are useful when optimizing blob usage per container.

Blob metadata is not included in this method. The Azure Blob Storage inventory feature uses the [List Blobs](#) REST API with default parameters. So, the example doesn't support snapshots, '\$' containers, and so on.

## Enable inventory reports

The first step in this method is to [enable inventory reports](#) on your storage account. You may have to wait up to 24 hours after enabling inventory reports for your first report to be generated.

When you have an inventory report to analyze, grant yourself read access to the container where the report CSV file resides by assigning yourself the **Storage Blob Data Reader** role. Be sure to use the email address of the account you're using to run the report. To learn how to assign an Azure role to a user with Azure role-based access control (Azure RBAC), follow the instructions provided in [Assign Azure roles using the Azure portal](#).

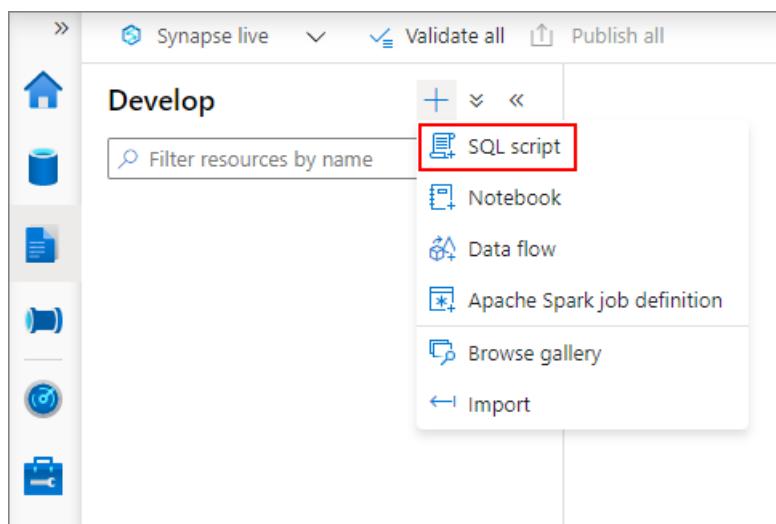
## Create an Azure Synapse workspace

Next, [create an Azure Synapse workspace](#) where you will execute a SQL query to report the inventory results.

## Create the SQL query

After you create your Azure Synapse workspace, do the following steps.

1. Navigate to <https://web.azuresynthesize.net>.
2. Select the **Develop** tab on the left edge.
3. Select the large plus sign (+) to add an item.
4. Select **SQL script**.



## Run the SQL query

1. Add the following SQL query in your Azure Synapse workspace to [read the inventory CSV file](#).

For the `bulk` parameter, use the URL of the inventory report CSV file that you want to analyze.

```
SELECT LEFT([Name], CHARINDEX('/', [Name]) - 1) AS Container,
 COUNT(*) AS TotalBlobCount,
 SUM([Content-Length]) AS TotalBlobSize
 FROM OPENROWSET(
 bulk '<URL to your inventory CSV file>',
 format='csv', parser_version='2.0', header_row=true
) AS Source
 GROUP BY LEFT([Name], CHARINDEX('/', [Name]) - 1)
```

2. Name your SQL query in the properties pane on the right.
3. Publish your SQL query by pressing CTRL+S or selecting the **Publish all** button.
4. Select the **Run** button to execute the SQL query. The blob count and total size per container are reported in the **Results** pane.

Container	TotalBlobCount	TotalBlobSize
inventory	1	978
cool	1	98641
test1	1	35938

## Next steps

- [Use Azure Storage blob inventory to manage blob data](#)
- [Calculate the total billing size of a blob container](#)

# Choose how to authorize access to blob data in the Azure portal

8/22/2022 • 6 minutes to read • [Edit Online](#)

When you access blob data using the [Azure portal](#), the portal makes requests to Azure Storage under the covers. A request to Azure Storage can be authorized using either your Azure AD account or the storage account access key. The portal indicates which method you are using, and enables you to switch between the two if you have the appropriate permissions.

You can also specify how to authorize an individual blob upload operation in the Azure portal. By default the portal uses whichever method you are already using to authorize a blob upload operation, but you have the option to change this setting when you upload a blob.

## Permissions needed to access blob data

Depending on how you want to authorize access to blob data in the Azure portal, you'll need specific permissions. In most cases, these permissions are provided via Azure role-based access control (Azure RBAC). For more information about Azure RBAC, see [What is Azure role-based access control \(Azure RBAC\)?](#)

### Use the account access key

To access blob data with the account access key, you must have an Azure role assigned to you that includes the Azure RBAC action **Microsoft.Storage/storageAccounts/listkeys/action**. This Azure role may be a built-in or a custom role. Built-in roles that support **Microsoft.Storage/storageAccounts/listkeys/action** include the following, in order from least to greatest permissions:

- The [Reader and Data Access](#) role
- The [Storage Account Contributor](#) role
- The Azure Resource Manager [Contributor](#) role
- The Azure Resource Manager [Owner](#) role

When you attempt to access blob data in the Azure portal, the portal first checks whether you have been assigned a role with **Microsoft.Storage/storageAccounts/listkeys/action**. If you have been assigned a role with this action, then the portal uses the account key for accessing blob data. If you have not been assigned a role with this action, then the portal attempts to access data using your Azure AD account.

#### IMPORTANT

When a storage account is locked with an Azure Resource Manager **ReadOnly** lock, the [List Keys](#) operation is not permitted for that storage account. [List Keys](#) is a POST operation, and all POST operations are prevented when a **ReadOnly** lock is configured for the account. For this reason, when the account is locked with a **ReadOnly** lock, users must use Azure AD credentials to access blob data in the portal. For information about accessing blob data in the portal with Azure AD, see [Use your Azure AD account](#).

## NOTE

The classic subscription administrator roles Service Administrator and Co-Administrator include the equivalent of the Azure Resource Manager [Owner](#) role. The [Owner](#) role includes all actions, including the [Microsoft.Storage/storageAccounts/listkeys/action](#), so a user with one of these administrative roles can also access blob data with the account key. For more information, see [Classic subscription administrator roles, Azure roles, and Azure AD administrator roles](#).

## Use your Azure AD account

To access blob data from the Azure portal using your Azure AD account, both of the following statements must be true for you:

- You have been assigned either a built-in or custom role that provides access to blob data.
- You have been assigned the Azure Resource Manager [Reader](#) role, at a minimum, scoped to the level of the storage account or higher. The [Reader](#) role grants the most restricted permissions, but another Azure Resource Manager role that grants access to storage account management resources is also acceptable.

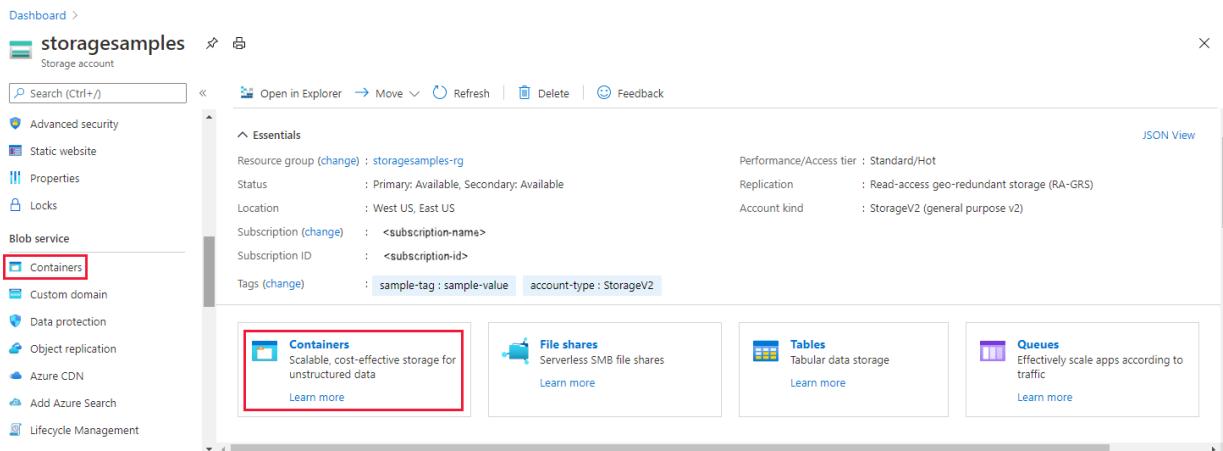
The Azure Resource Manager [Reader](#) role permits users to view storage account resources, but not modify them. It does not provide read permissions to data in Azure Storage, but only to account management resources. The [Reader](#) role is necessary so that users can navigate to blob containers in the Azure portal.

For information about the built-in roles that support access to blob data, see [Authorize access to blobs using Azure Active Directory](#).

Custom roles can support different combinations of the same permissions provided by the built-in roles. For more information about creating Azure custom roles, see [Azure custom roles](#) and [Understand role definitions for Azure resources](#).

## Navigate to blobs in the Azure portal

To view blob data in the portal, navigate to the [Overview](#) for your storage account, and click on the links for [Blobs](#). Alternatively you can navigate to the [Containers](#) section in the menu.



## Determine the current authentication method

When you navigate to a container, the Azure portal indicates whether you are currently using the account access key or your Azure AD account to authenticate.

### Authenticate with the account access key

If you are authenticating using the account access key, you'll see [Access Key](#) specified as the authentication method in the portal:

The screenshot shows the Azure Storage Container Overview page for a container named 'container'. The left sidebar includes 'Overview', 'Access Control (IAM)', 'Settings' (with 'Access policy', 'Properties', 'Metadata', and 'Editor (preview)'), and a search bar. The main area displays the 'Authentication method' as 'Access key (Switch to Azure AD User Account)' and the 'Location' as 'container'. A search bar at the top right allows searching by prefix (case-sensitive). Below it is a table with columns: NAME, MODIFIED, ACCESS TIER, BLOB TYPE, SIZE, and LEASE STATE. A single row is shown for 'testfile.txt', which was modified on 2/28/2019, 3:48:15 PM, is in the Hot (Inferred) tier, is a Block blob, has a size of 125 B, and is in an Available lease state. The 'NAME' column for this row is highlighted with a red box.

To switch to using Azure AD account, click the link highlighted in the image. If you have the appropriate permissions via the Azure roles that are assigned to you, you'll be able to proceed. However, if you lack the right permissions, you'll see an error message like the following one:

The screenshot shows the same Azure Storage Container Overview page as before, but with a different authentication method. The 'Authentication method' is now 'Azure AD User Account (Switch to Access key)' and the 'Location' is 'container'. A prominent error message in a red box states: 'You do not have permissions to list the data using your user account with Azure AD. Click to learn more about authenticating with Azure AD.' Below this, the table shows 'No blobs found.' The 'NAME' column for the previous entry ('testfile.txt') is highlighted with a red box.

Notice that no blobs appear in the list if your Azure AD account lacks permissions to view them. Click on the **Switch to access key** link to use the access key for authentication again.

### Authenticate with your Azure AD account

If you are authenticating using your Azure AD account, you'll see **Azure AD User Account** specified as the authentication method in the portal:

The screenshot shows the same Azure Storage Container Overview page again, but this time the 'Authentication method' is correctly set to 'Azure AD User Account (Switch to Access key)'. The 'NAME' column for the 'testfile.txt' blob is highlighted with a red box, indicating it is now visible due to successful authentication.

To switch to using the account access key, click the link highlighted in the image. If you have access to the account key, then you'll be able to proceed. However, if you lack access to the account key, you'll see an error message like the following one:

The screenshot shows the Azure Storage Container Overview page for a container named 'container'. The left sidebar includes options like Overview, Access Control (IAM), Settings (Access policy, Properties, Metadata, Editor (preview)), and a search bar. The main area has a red warning banner stating 'You do not have permissions to use the access key to list data. Click to learn more about authenticating with Azure Storage.' Below this, the 'Authentication method' is set to 'Access key (Switch to Azure AD User Account)'. A table header for blob listing includes columns: NAME, MODIFIED, ACCESS TIER, BLOB TYPE, SIZE, and LEASE STATE. A message 'No blobs found.' is displayed below the table.

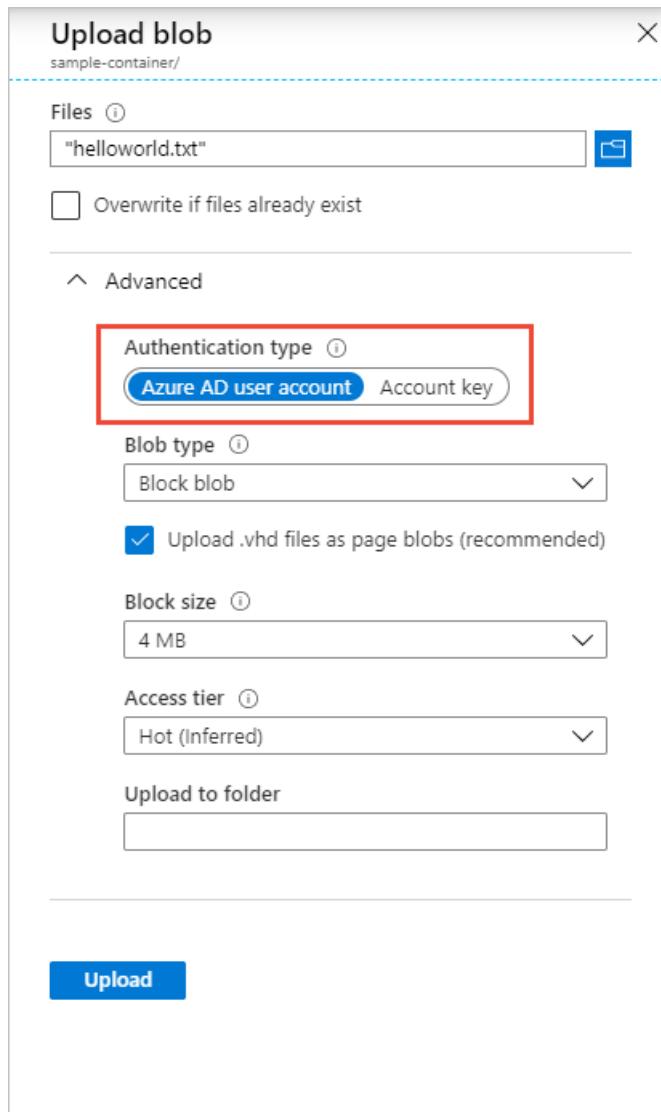
Notice that no blobs appear in the list if you do not have access to the account keys. Click on the **Switch to Azure AD User Account** link to use your Azure AD account for authentication again.

## Specify how to authorize a blob upload operation

When you upload a blob from the Azure portal, you can specify whether to authenticate and authorize that operation with the account access key or with your Azure AD credentials. By default, the portal uses the current authentication method, as shown in [Determine the current authentication method](#).

To specify how to authorize a blob upload operation, follow these steps:

1. In the Azure portal, navigate to the container where you wish to upload a blob.
2. Select the **Upload** button.
3. Expand the **Advanced** section to display the advanced properties for the blob.
4. In the **Authentication Type** field, indicate whether you want to authorize the upload operation by using your Azure AD account or with the account access key, as shown in the following image:



## Default to Azure AD authorization in the Azure portal

When you create a new storage account, you can specify that the Azure portal will default to authorization with Azure AD when a user navigates to blob data. You can also configure this setting for an existing storage account. This setting specifies the default authorization method only, so keep in mind that a user can override this setting and choose to authorize data access with the account key.

To specify that the portal will use Azure AD authorization by default for data access when you create a storage account, follow these steps:

1. Create a new storage account, following the instructions in [Create a storage account](#).
2. On the **Advanced** tab, in the **Security** section, check the box next to **Default to Azure Active Directory authorization in the Azure portal**.

## Create a storage account ...

[Basics](#) [Advanced](#) [Networking](#) [Data protection](#) [Tags](#) [Review + create](#)**Security**

Configure security settings that impact your storage account.

Require secure transfer for REST API operations  ⓘEnable infrastructure encryption  ⓘEnable blob public access  ⓘEnable storage account key access  ⓘDefault to Azure Active Directory authorization in the Azure portal  ⓘMinimum TLS version  ⓘ[Review + create](#)

&lt; Previous

Next : Networking &gt;

3. Select the **Review + create** button to run validation and create the account.

To update this setting for an existing storage account, follow these steps:

1. Navigate to the account overview in the Azure portal.
2. Under **Settings**, select **Configuration**.
3. Set **Default to Azure Active Directory authorization in the Azure portal** to **Enabled**.

Default to Azure Active Directory authorization in the Azure portal ⓘ

 Disabled  Enabled

## Next steps

- [Authorize access to data in Azure Storage](#)
- [Assign an Azure role for access to blob data](#)

# Run PowerShell commands with Azure AD credentials to access blob data

8/22/2022 • 3 minutes to read • [Edit Online](#)

Azure Storage provides extensions for PowerShell that enable you to sign in and run scripting commands with Azure Active Directory (Azure AD) credentials. When you sign in to PowerShell with Azure AD credentials, an OAuth 2.0 access token is returned. That token is automatically used by PowerShell to authorize subsequent data operations against Blob storage. For supported operations, you no longer need to pass an account key or SAS token with the command.

You can assign permissions to blob data to an Azure AD security principal via Azure role-based access control (Azure RBAC). For more information about Azure roles in Azure Storage, see [Assign an Azure role for access to blob data](#).

## Supported operations

The Azure Storage extensions are supported for operations on blob data. Which operations you may call depends on the permissions granted to the Azure AD security principal with which you sign in to PowerShell. Permissions to Azure Storage containers are assigned via Azure RBAC. For example, if you have been assigned the **Blob Data Reader** role, then you can run scripting commands that read data from a container. If you have been assigned the **Blob Data Contributor** role, then you can run scripting commands that read, write, or delete a container or the data they contain.

For details about the permissions required for each Azure Storage operation on a container, see [Call storage operations with OAuth tokens](#).

### IMPORTANT

When a storage account is locked with an Azure Resource Manager **ReadOnly** lock, the **List Keys** operation is not permitted for that storage account. **List Keys** is a POST operation, and all POST operations are prevented when a **ReadOnly** lock is configured for the account. For this reason, when the account is locked with a **ReadOnly** lock, users who do not already possess the account keys must use Azure AD credentials to access blob data. In PowerShell, include the `-UseConnectedAccount` parameter to create an **AzureStorageContext** object with your Azure AD credentials.

## Call PowerShell commands using Azure AD credentials

To use Azure PowerShell to sign in and run subsequent operations against Azure Storage using Azure AD credentials, create a storage context to reference the storage account, and include the `-UseConnectedAccount` parameter.

The following example shows how to create a container in a new storage account from Azure PowerShell using your Azure AD credentials. Remember to replace placeholder values in angle brackets with your own values:

1. Sign in to your Azure account with the [Connect-AzAccount](#) command:

```
Connect-AzAccount
```

For more information about signing into Azure with PowerShell, see [Sign in with Azure PowerShell](#).

2. Create an Azure resource group by calling [New-AzResourceGroup](#).

```
$resourceGroup = "sample-resource-group-ps"
.setLocation = "eastus"
New-AzResourceGroup -Name $resourceGroup -Location $location
```

3. Create a storage account by calling [New-AzStorageAccount](#).

```
$storageAccount = New-AzStorageAccount -ResourceGroupName $resourceGroup `
-Name "<storage-account>" `
-SkuName Standard_LRS `
-Location $location `
```

4. Get the storage account context that specifies the new storage account by calling [New-AzStorageContext](#).

When acting on a storage account, you can reference the context instead of repeatedly passing in the credentials. Include the `-UseConnectedAccount` parameter to call any subsequent data operations using your Azure AD credentials:

```
$ctx = New-AzStorageContext -StorageAccountName "<storage-account>" -UseConnectedAccount
```

5. Before you create the container, assign the [Storage Blob Data Contributor](#) role to yourself. Even though you are the account owner, you need explicit permissions to perform data operations against the storage account. For more information about assigning Azure roles, see [Assign an Azure role for access to blob data](#).

**IMPORTANT**

Azure role assignments may take a few minutes to propagate.

6. Create a container by calling [New-AzStorageContainer](#). Because this call uses the context created in the previous steps, the container is created using your Azure AD credentials.

```
$containerName = "sample-container"
New-AzStorageContainer -Name $containerName -Context $ctx
```

## Next steps

- [Assign an Azure role for access to blob data](#)
- [Authorize access to blob data with managed identities for Azure resources](#)

# Choose how to authorize access to blob data with Azure CLI

8/22/2022 • 6 minutes to read • [Edit Online](#)

Azure Storage provides extensions for Azure CLI that enable you to specify how you want to authorize operations on blob data. You can authorize data operations in the following ways:

- With an Azure Active Directory (Azure AD) security principal. Microsoft recommends using Azure AD credentials for superior security and ease of use.
- With the account access key or a shared access signature (SAS) token.

## Specify how data operations are authorized

Azure CLI commands for reading and writing blob data include the optional `--auth-mode` parameter. Specify this parameter to indicate how a data operation is to be authorized:

- Set the `--auth-mode` parameter to `login` to sign in using an Azure AD security principal (recommended).
- Set the `--auth-mode` parameter to the legacy `key` value to attempt to retrieve the account access key to use for authorization. If you omit the `--auth-mode` parameter, then the Azure CLI also attempts to retrieve the access key.

To use the `--auth-mode` parameter, make sure that you have installed Azure CLI version 2.0.46 or later. Run `az --version` to check your installed version.

### NOTE

When a storage account is locked with an Azure Resource Manager **ReadOnly** lock, the [List Keys](#) operation is not permitted for that storage account. **List Keys** is a POST operation, and all POST operations are prevented when a **ReadOnly** lock is configured for the account. For this reason, when the account is locked with a **ReadOnly** lock, users who do not already possess the account keys must use Azure AD credentials to access blob data.

### IMPORTANT

If you omit the `--auth-mode` parameter or set it to `key`, then the Azure CLI attempts to use the account access key for authorization. In this case, Microsoft recommends that you provide the access key either on the command or in the **AZURE\_STORAGE\_KEY** environment variable. For more information about environment variables, see the section titled [Set environment variables for authorization parameters](#).

If you do not provide the access key, then the Azure CLI attempts to call the Azure Storage resource provider to retrieve it for each operation. Performing many data operations that require a call to the resource provider may result in throttling. For more information about resource provider limits, see [Scalability and performance targets for the Azure Storage resource provider](#).

## Authorize with Azure AD credentials

When you sign in to Azure CLI with Azure AD credentials, an OAuth 2.0 access token is returned. That token is automatically used by Azure CLI to authorize subsequent data operations against Blob or Queue storage. For supported operations, you no longer need to pass an account key or SAS token with the command.

You can assign permissions to blob data to an Azure AD security principal via Azure role-based access control (Azure RBAC). For more information about Azure roles in Azure Storage, see [Assign an Azure role for access to blob data](#).

## Permissions for calling data operations

The Azure Storage extensions are supported for operations on blob data. Which operations you may call depends on the permissions granted to the Azure AD security principal with which you sign in to Azure CLI. Permissions to Azure Storage containers are assigned via Azure RBAC. For example, if you are assigned the **Storage Blob Data Reader** role, then you can run scripting commands that read data from a container. If you are assigned the **Storage Blob Data Contributor** role, then you can run scripting commands that read, write, or delete a container or the data it contains.

For details about the permissions required for each Azure Storage operation on a container, see [Call storage operations with OAuth tokens](#).

### Example: Authorize an operation to create a container with Azure AD credentials

The following example shows how to create a container from Azure CLI using your Azure AD credentials. To create the container, you'll need to sign in to the Azure CLI, and you'll need a resource group and a storage account. To learn how to create these resources, see [Quickstart: Create, download, and list blobs with Azure CLI](#).

1. Before you create the container, assign the **Storage Blob Data Contributor** role to yourself. Even though you are the account owner, you need explicit permissions to perform data operations against the storage account. For more information about assigning Azure roles, see [Assign an Azure role for access to blob data](#).

#### IMPORTANT

Azure role assignments may take a few minutes to propagate.

2. Call the `az storage container create` command with the `--auth-mode` parameter set to `login` to create the container using your Azure AD credentials. Remember to replace placeholder values in angle brackets with your own values:

```
az storage container create \
 --account-name <storage-account> \
 --name sample-container \
 --auth-mode login
```

## Authorize with the account access key

If you possess the account key, you can call any Azure Storage data operation. In general, using the account key is less secure. If the account key is compromised, all data in your account may be compromised.

The following example shows how to create a container using the account access key. Specify the account key, and provide the `--auth-mode` parameter with the `key` value:

```
az storage container create \
 --account-name <storage-account> \
 --name sample-container \
 --account-key <key>
 --auth-mode key
```

## IMPORTANT

When a storage account is locked with an Azure Resource Manager **ReadOnly** lock, the [List Keys](#) operation is not permitted for that storage account. **List Keys** is a POST operation, and all POST operations are prevented when a **ReadOnly** lock is configured for the account. For this reason, when the account is locked with a **ReadOnly** lock, users must access data with Azure AD credentials.

## Authorize with a SAS token

If you possess a SAS token, you can call data operations that are permitted by the SAS. The following example shows how to create a container using a SAS token:

```
az storage container create \
--account-name <storage-account> \
--name sample-container \
--sas-token <token>
```

## Set environment variables for authorization parameters

You can specify authorization parameters in environment variables to avoid including them on every call to an Azure Storage data operation. The following table describes the available environment variables.

ENVIRONMENT VARIABLE	DESCRIPTION
AZURE_STORAGE_ACCOUNT	The storage account name. This variable should be used in conjunction with either the storage account key or a SAS token. If neither are present, the Azure CLI attempts to retrieve the storage account access key by using the authenticated Azure AD account. If a large number of commands are executed at one time, the Azure Storage resource provider throttling limit may be reached. For more information about resource provider limits, see <a href="#">Scalability and performance targets for the Azure Storage resource provider</a> .
AZURE_STORAGE_KEY	The storage account key. This variable must be used in conjunction with the storage account name.
AZURE_STORAGE_CONNECTION_STRING	A connection string that includes the storage account key or a SAS token. This variable must be used in conjunction with the storage account name.
AZURE_STORAGE_SAS_TOKEN	A shared access signature (SAS) token. This variable must be used in conjunction with the storage account name.
AZURE_STORAGE_AUTH_MODE	The authorization mode with which to run the command. Permitted values are <code>login</code> (recommended) or <code>key</code> . If you specify <code>login</code> , the Azure CLI uses your Azure AD credentials to authorize the data operation. If you specify the legacy <code>key</code> mode, the Azure CLI attempts to query for the account access key and to authorize the command with the key.

## Next steps

- Assign an Azure role for access to blob data
- Authorize access to blob data with managed identities for Azure resources

# Assign an Azure role for access to blob data

8/22/2022 • 7 minutes to read • [Edit Online](#)

Azure Active Directory (AAD) authorizes access rights to secured resources through [Azure role-based access control \(Azure RBAC\)](#). Azure Storage defines a set of Azure built-in roles that encompass common sets of permissions used to access blob data.

When an Azure role is assigned to an Azure AD security principal, Azure grants access to those resources for that security principal. An Azure AD security principal may be a user, a group, an application service principal, or a [managed identity for Azure resources](#).

To learn more about using Azure AD to authorize access to blob data, see [Authorize access to blobs using Azure Active Directory](#).

## NOTE

This article shows how to assign an Azure role for access to blob data in a storage account. To learn about assigning roles for management operations in Azure Storage, see [Use the Azure Storage resource provider to access management resources](#).

## Assign an Azure role

You can use the Azure portal, PowerShell, Azure CLI, or an Azure Resource Manager template to assign a role for data access.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Template](#)

To access blob data in the Azure portal with Azure AD credentials, a user must have the following role assignments:

- A data access role, such as **Storage Blob Data Reader** or **Storage Blob Data Contributor**
- The Azure Resource Manager **Reader** role, at a minimum

To learn how to assign these roles to a user, follow the instructions provided in [Assign Azure roles using the Azure portal](#).

The **Reader** role is an Azure Resource Manager role that permits users to view storage account resources, but not modify them. It does not provide read permissions to data in Azure Storage, but only to account management resources. The **Reader** role is necessary so that users can navigate to blob containers in the Azure portal.

For example, if you assign the **Storage Blob Data Contributor** role to user Mary at the level of a container named **sample-container**, then Mary is granted read, write, and delete access to all of the blobs in that container. However, if Mary wants to view a blob in the Azure portal, then the **Storage Blob Data Contributor** role by itself will not provide sufficient permissions to navigate through the portal to the blob in order to view it. The additional permissions are required to navigate through the portal and view the other resources that are visible there.

A user must be assigned the **Reader** role to use the Azure portal with Azure AD credentials. However, if a user has been assigned a role with **Microsoft.Storage/storageAccounts/listKeys/action** permissions, then the user can use the portal with the storage account keys, via Shared Key authorization. To use the storage account keys, Shared Key access must be permitted for the storage account. For more information on permitting or disallowing Shared Key access, see [Prevent Shared Key authorization for an Azure Storage account](#).

You can also assign an Azure Resource Manager role that provides additional permissions beyond than the **Reader** role. Assigning the least possible permissions is recommended as a security best practice. For more information, see [Best practices for Azure RBAC](#).

**NOTE**

Prior to assigning yourself a role for data access, you will be able to access data in your storage account via the Azure portal because the Azure portal can also use the account key for data access. For more information, see [Choose how to authorize access to blob data in the Azure portal](#).

Keep in mind the following points about Azure role assignments in Azure Storage:

- When you create an Azure Storage account, you are not automatically assigned permissions to access data via Azure AD. You must explicitly assign yourself an Azure role for Azure Storage. You can assign it at the level of your subscription, resource group, storage account, or container.
- If the storage account is locked with an Azure Resource Manager read-only lock, then the lock prevents the assignment of Azure roles that are scoped to the storage account or a container.
- If you have set the appropriate allow permissions to access data via Azure AD and are unable to access the data, for example you are getting an "AuthorizationPermissionMismatch" error. Be sure to allow enough time for the permissions changes you have made in Azure AD to replicate, and be sure that you do not have any deny assignments that block your access, see [Understand Azure deny assignments](#).

**NOTE**

You can create custom Azure RBAC roles for granular access to blob data. For more information, see [Azure custom roles](#).

## Next steps

- [What is Azure role-based access control \(Azure RBAC\)?](#)
- [Best practices for Azure RBAC](#)

# Manage storage account access keys

8/22/2022 • 11 minutes to read • [Edit Online](#)

When you create a storage account, Azure generates two 512-bit storage account access keys for that account. These keys can be used to authorize access to data in your storage account via Shared Key authorization.

Microsoft recommends that you use Azure Key Vault to manage your access keys, and that you regularly rotate and regenerate your keys. Using Azure Key Vault makes it easy to rotate your keys without interruption to your applications. You can also manually rotate your keys.

## Protect your access keys

Your storage account access keys are similar to a root password for your storage account. Always be careful to protect your access keys. Use Azure Key Vault to manage and rotate your keys securely. Avoid distributing access keys to other users, hard-coding them, or saving them anywhere in plain text that is accessible to others. Rotate your keys if you believe they may have been compromised.

### NOTE

Microsoft recommends using Azure Active Directory (Azure AD) to authorize requests against blob and queue data if possible, rather than using the account keys (Shared Key authorization). Authorization with Azure AD provides superior security and ease of use over Shared Key authorization.

To protect an Azure Storage account with Azure AD Conditional Access policies, you must disallow Shared Key authorization for the storage account. For more information about how to disallow Shared Key authorization, see [Prevent Shared Key authorization for an Azure Storage account](#).

## View account access keys

You can view and copy your account access keys with the Azure portal, PowerShell, or Azure CLI. The Azure portal also provides a connection string for your storage account that you can copy.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To view and copy your storage account access keys or connection string from the Azure portal:

1. In the [Azure portal](#), go to your storage account.
2. Under **Security + networking**, select **Access keys**. Your account access keys appear, as well as the complete connection string for each key.
3. Select **Show keys** to show your access keys and connection strings and to enable buttons to copy the values.
4. Under **key1**, find the **Key** value. Select the **Copy** button to copy the account key.
5. Alternately, you can copy the entire connection string. Under **key1**, find the **Connection string** value. Select the **Copy** button to copy the connection string.



You can use either of the two keys to access Azure Storage, but in general it's a good practice to use the first key, and reserve the use of the second key for when you are rotating keys.

To view or read an account's access keys, the user must either be a Service Administrator, or must be assigned an Azure role that includes the `Microsoft.Storage/storageAccounts/listkeys/action`. Some Azure built-in roles that include this action are the **Owner**, **Contributor**, and **Storage Account Key Operator Service Role** roles. For more information about the Service Administrator role, see [Classic subscription administrator roles](#), [Azure roles](#), and [Azure AD roles](#). For detailed information about built-in roles for Azure Storage, see the **Storage** section in [Azure built-in roles for Azure RBAC](#).

## Use Azure Key Vault to manage your access keys

Microsoft recommends using Azure Key Vault to manage and rotate your access keys. Your application can securely access your keys in Key Vault, so that you can avoid storing them with your application code. For more information about using Key Vault for key management, see the following articles:

- [Manage storage account keys with Azure Key Vault and PowerShell](#)
- [Manage storage account keys with Azure Key Vault and the Azure CLI](#)

## Manually rotate access keys

Microsoft recommends that you rotate your access keys periodically to help keep your storage account secure. If possible, use Azure Key Vault to manage your access keys. If you are not using Key Vault, you will need to rotate your keys manually.

Two access keys are assigned so that you can rotate your keys. Having two keys ensures that your application maintains access to Azure Storage throughout the process.

### WARNING

Regenerating your access keys can affect any applications or Azure services that are dependent on the storage account key. Any clients that use the account key to access the storage account must be updated to use the new key, including media services, cloud, desktop and mobile applications, and graphical user interface applications for Azure Storage, such as [Azure Storage Explorer](#).

If you plan to manually rotate access keys, Microsoft recommends that you set a key expiration policy. For more information, see [Create a key expiration policy](#).

After you create the key expiration policy, you can use Azure Policy to monitor whether a storage account's keys have been rotated within the recommended interval. For details, see [Check for key expiration policy violations](#).

- [Portal](#)

- [PowerShell](#)
- [Azure CLI](#)

To rotate your storage account access keys in the Azure portal:

1. Update the connection strings in your application code to reference the secondary access key for the storage account.
2. Navigate to your storage account in the [Azure portal](#).
3. Under **Security + networking**, select **Access keys**.
4. To regenerate the primary access key for your storage account, select the **Regenerate** button next to the primary access key.
5. Update the connection strings in your code to reference the new primary access key.
6. Regenerate the secondary access key in the same manner.

**Caution**

Microsoft recommends using only one of the keys in all of your applications at the same time. If you use Key 1 in some places and Key 2 in others, you will not be able to rotate your keys without some application losing access.

To rotate an account's access keys, the user must either be a Service Administrator, or must be assigned an Azure role that includes the `Microsoft.Storage/storageAccounts/regeneratekey/action`. Some Azure built-in roles that include this action are the **Owner**, **Contributor**, and **Storage Account Key Operator Service Role** roles. For more information about the Service Administrator role, see [Classic subscription administrator roles, Azure roles, and Azure AD roles](#). For detailed information about Azure built-in roles for Azure Storage, see the **Storage** section in [Azure built-in roles for Azure RBAC](#).

## Create a key expiration policy

A key expiration policy enables you to set a reminder for the rotation of the account access keys. The reminder is displayed if the specified interval has elapsed and the keys have not yet been rotated. After you create a key expiration policy, you can monitor your storage accounts for compliance to ensure that the account access keys are rotated regularly.

**NOTE**

Before you can create a key expiration policy, you may need to rotate each of your account access keys at least once.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To create a key expiration policy in the Azure portal:

1. In the [Azure portal](#), go to your storage account.
2. Under **Security + networking**, select **Access keys**. Your account access keys appear, as well as the complete connection string for each key.
3. Select the **Set rotation reminder** button. If the **Set rotation reminder** button is grayed out, you will need to rotate each of your keys. Follow the steps described in [Manually rotate access keys](#) to rotate the keys.
4. In **Set a reminder to rotate access keys**, select the **Enable key rotation reminders** checkbox and set a frequency for the reminder.
5. Select **Save**.

The screenshot shows the Azure Storage account 'keyrotationdocs' with the 'Access keys' blade open. In the top right, a modal window titled 'Set a reminder to rotate access keys' is displayed. The modal contains instructions about manually rotating keys and has a checked checkbox labeled 'Enable key rotation reminders'. Below this are fields for 'Send reminders' (set to 'Custom') and 'Remind me every' (set to '60 Days'). At the bottom of the modal are 'Save' and 'Cancel' buttons. The main pane shows two keys: 'key1' and 'key2', both last rotated on 10/4/2021. Each key has a 'Rotate key' button and a 'Connection string' section.

## Check for key expiration policy violations

You can monitor your storage accounts with Azure Policy to ensure that account access keys have been rotated within the recommended period. Azure Storage provides a built-in policy for ensuring that storage account access keys are not expired. For more information about the built-in policy, see **Storage account keys should not be expired** in [List of built-in policy definitions](#).

### Assign the built-in policy for a resource scope

Follow these steps to assign the built-in policy to the appropriate scope in the Azure portal:

1. In the Azure portal, search for *Policy* to display the Azure Policy dashboard.
2. In the **Authoring** section, select **Assignments**.
3. Choose **Assign policy**.
4. On the **Basics** tab of the **Assign policy** page, in the **Scope** section, specify the scope for the policy assignment. Select the **More** button to choose the subscription and optional resource group.
5. For the **Policy definition** field, select the **More** button, and enter *storage account keys* in the **Search** field. Select the policy definition named **Storage account keys should not be expired**.

## Available Definitions

X

Type

Search

### Policy Definitions (1)

#### Storage account keys should not be expired

Built-in

Ensure the user storage account keys are not expired when key expiration policy is set, for improving security of account keys by taking action when the keys are expired.

6. Select **Review + create** to assign the policy definition to the specified scope.

Dashboard > Policy >

## Assign policy

### Scope

Scope [Learn more about setting the scope \\*](#)



### Exclusions



### Basics

Policy definition \*



Assignment name \* [\(i\)](#)



### Description

Policy enforcement [\(i\)](#)

Assigned by

## Monitor compliance with the key expiration policy

To monitor your storage accounts for compliance with the key expiration policy, follow these steps:

1. On the Azure Policy dashboard, locate the built-in policy definition for the scope that you specified in the policy assignment. You can search for *Storage account keys should not be expired* in the **Search** box to filter for the built-in policy.
2. Select the policy name with the desired scope.
3. On the **Policy assignment** page for the built-in policy, select **View compliance**. Any storage accounts in the specified subscription and resource group that do not meet the policy requirements appear in the compliance report.

**Storage account keys should not be expired**

**Selected Scopes**: 2 selected subscriptions

**Compliance state**: Non-compliant

**Overall resource compliance**: 99% (134 out of 135)

**Resources by compliance state**:

- 134 - Compliant
- 0 - Exempt
- 1 - Non-compliant

**Details**

- Effect Type: Audit
- Parent Initiative: <>NONE<>

Name	Compliance state	Compliance reason	Resource Type	Location	Scope	Last evaluated
keyexpiry	Non-compliant	Details	microsoft.storage/storageAccounts	West US 2	Azure Storage content development and testing/storagesamples-rg	12/9/2021, 6:34 AM

To bring a storage account into compliance, rotate the account access keys.

## Next steps

- [Azure storage account overview](#)
- [Create a storage account](#)
- [Prevent Shared Key authorization for an Azure Storage account](#)

# Configure Azure Storage connection strings

8/22/2022 • 7 minutes to read • [Edit Online](#)

A connection string includes the authorization information required for your application to access data in an Azure Storage account at runtime using Shared Key authorization. You can configure connection strings to:

- Connect to the Azurite storage emulator.
- Access a storage account in Azure.
- Access specified resources in Azure via a shared access signature (SAS).

To learn how to view your account access keys and copy a connection string, see [Manage storage account access keys](#).

## Protect your access keys

Your storage account access keys are similar to a root password for your storage account. Always be careful to protect your access keys. Use Azure Key Vault to manage and rotate your keys securely. Avoid distributing access keys to other users, hard-coding them, or saving them anywhere in plain text that is accessible to others. Rotate your keys if you believe they may have been compromised.

### NOTE

Microsoft recommends using Azure Active Directory (Azure AD) to authorize requests against blob and queue data if possible, rather than using the account keys (Shared Key authorization). Authorization with Azure AD provides superior security and ease of use over Shared Key authorization.

To protect an Azure Storage account with Azure AD Conditional Access policies, you must disallow Shared Key authorization for the storage account. For more information about how to disallow Shared Key authorization, see [Prevent Shared Key authorization for an Azure Storage account](#).

## Store a connection string

Your application needs to access the connection string at runtime to authorize requests made to Azure Storage. You have several options for storing your connection string:

- You can store your connection string in an environment variable.
- An application running on the desktop or on a device can store the connection string in an **app.config** or **web.config** file. Add the connection string to the **AppSettings** section in these files.
- An application running in an Azure cloud service can store the connection string in the [Azure service configuration schema \(.cscfg\) file](#). Add the connection string to the **ConfigurationSettings** section of the service configuration file.

Storing your connection string in a configuration file makes it easy to update the connection string to switch between the [Azurite storage emulator](#) and an Azure storage account in the cloud. You only need to edit the connection string to point to your target environment.

You can use the [Microsoft Azure Configuration Manager](#) to access your connection string at runtime regardless of where your application is running.

## Configure a connection string for Azurite

The emulator supports a single fixed account and a well-known authentication key for Shared Key authentication. This account and key are the only Shared Key credentials permitted for use with the emulator. They are:

```
Account name: devstoreaccount1
Account key: Eby8vdM02xN0cqFlqUwJPLlEt1CDXJ10UzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==
```

#### NOTE

The authentication key supported by the emulator is intended only for testing the functionality of your client authentication code. It does not serve any security purpose. You cannot use your production storage account and key with the emulator. You should not use the development account with production data.

The emulator supports connection via HTTP only. However, HTTPS is the recommended protocol for accessing resources in a production Azure storage account.

#### Connect to the emulator account using the shortcut

The easiest way to connect to the emulator from your application is to configure a connection string in your application's configuration file that references the shortcut `UseDevelopmentStorage=true`. The shortcut is equivalent to the full connection string for the emulator, which specifies the account name, the account key, and the emulator endpoints for each of the Azure Storage services:

```
DefaultEndpointsProtocol=http;AccountName=devstoreaccount1;
AccountKey=Eby8vdM02xN0cqFlqUwJPLlEt1CDXJ10UzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==;
BlobEndpoint=http://127.0.0.1:10000/devstoreaccount1;
QueueEndpoint=http://127.0.0.1:10001/devstoreaccount1;
TableEndpoint=http://127.0.0.1:10002/devstoreaccount1;
```

The following .NET code snippet shows how you can use the shortcut from a method that takes a connection string. For example, the `BlobContainerClient(String, String)` constructor takes a connection string.

```
BlobContainerClient blobContainerClient = new BlobContainerClient("UseDevelopmentStorage=true", "sample-
container");
blobContainerClient.CreateIfNotExists();
```

Make sure that the emulator is running before calling the code in the snippet.

For more information about Azurite, see [Use the Azurite emulator for local Azure Storage development](#).

## Configure a connection string for an Azure storage account

To create a connection string for your Azure storage account, use the following format. Indicate whether you want to connect to the storage account through HTTPS (recommended) or HTTP, replace `myAccountName` with the name of your storage account, and replace `myAccountKey` with your account access key:

```
DefaultEndpointsProtocol=[http|https];AccountName=myAccountName;AccountKey=myAccountKey
```

For example, your connection string might look similar to:

```
DefaultEndpointsProtocol=https;AccountName=storagesample;AccountKey=<account-key>
```

Although Azure Storage supports both HTTP and HTTPS in a connection string, *HTTPS is highly recommended*.

**TIP**

You can find your storage account's connection strings in the [Azure portal](#). Navigate to **SETTINGS > Access keys** in your storage account's menu blade to see connection strings for both primary and secondary access keys.

## Create a connection string using a shared access signature

If you possess a shared access signature (SAS) URL that grants you access to resources in a storage account, you can use the SAS in a connection string. Because the SAS contains the information required to authenticate the request, a connection string with a SAS provides the protocol, the service endpoint, and the necessary credentials to access the resource.

To create a connection string that includes a shared access signature, specify the string in the following format:

```
BlobEndpoint=myBlobEndpoint;
QueueEndpoint=myQueueEndpoint;
TableEndpoint=myTableEndpoint;
FileEndpoint=myFileEndpoint;
SharedAccessSignature=sasToken
```

Each service endpoint is optional, although the connection string must contain at least one.

**NOTE**

Using HTTPS with a SAS is recommended as a best practice.

If you are specifying a SAS in a connection string in a configuration file, you may need to encode special characters in the URL.

### Service SAS example

Here's an example of a connection string that includes a service SAS for Blob storage:

```
BlobEndpoint=https://storagesample.blob.core.windows.net;
SharedAccessSignature=sv=2015-04-05&sr=b&si=tutorial-policy-
635959936145100803&sig=9aCzs76n0E7y5BpEi2GvsSv433BZa22leD0ZXX%2BXXIU%3D
```

And here's an example of the same connection string with encoding of special characters:

```
BlobEndpoint=https://storagesample.blob.core.windows.net;
SharedAccessSignature=sv=2015-04-05&sr=b&si=tutorial-policy-
635959936145100803&sig=9aCzs76n0E7y5BpEi2GvsSv433BZa22leD0ZXX%2BXXIU%3D
```

### Account SAS example

Here's an example of a connection string that includes an account SAS for Blob and File storage. Note that endpoints for both services are specified:

```
BlobEndpoint=https://storagesample.blob.core.windows.net;
FileEndpoint=https://storagesample.file.core.windows.net;
SharedAccessSignature=sv=2015-07-
08&sig=iCvQmdZngZNW%2F4vw43j6%2BVz6fndHF5LI639QJba4r8o%3D&spr=https&st=2016-04-12T03%3A24%3A31Z&se=2016-04-
13T03%3A29%3A31Z&srt=s&ss=bf&sp=rwl
```

And here's an example of the same connection string with URL encoding:

```
BlobEndpoint=https://storagesample.blob.core.windows.net;
FileEndpoint=https://storagesample.file.core.windows.net;
SharedAccessSignature=sv=2015-07-
08&sig=iCvQmdZngZNW%2F4vw43j6%2BVz6fndHF5LI639QJba4r8o%3D&spr=https&st=2016-04-
12T03%3A24%3A31Z&se=2016-04-13T03%3A29%3A31Z&srt=s&ss=bf&sp=rwl
```

## Create a connection string for an explicit storage endpoint

You can specify explicit service endpoints in your connection string instead of using the default endpoints. To create a connection string that specifies an explicit endpoint, specify the complete service endpoint for each service, including the protocol specification (HTTPS (recommended) or HTTP), in the following format:

```
DefaultEndpointsProtocol=[http|https];
BlobEndpoint=myBlobEndpoint;
FileEndpoint=myFileEndpoint;
QueueEndpoint=myQueueEndpoint;
TableEndpoint=myTableEndpoint;
AccountName=myAccountName;
AccountKey=myAccountKey
```

One scenario where you might wish to specify an explicit endpoint is when you've mapped your Blob storage endpoint to a [custom domain](#). In that case, you can specify your custom endpoint for Blob storage in your connection string. You can optionally specify the default endpoints for the other services if your application uses them.

Here is an example of a connection string that specifies an explicit endpoint for the Blob service:

```
Blob endpoint only
DefaultEndpointsProtocol=https;
BlobEndpoint=http://www.mydomain.com;
AccountName=storagesample;
AccountKey=<account-key>
```

This example specifies explicit endpoints for all services, including a custom domain for the Blob service:

```
All service endpoints
DefaultEndpointsProtocol=https;
BlobEndpoint=http://www.mydomain.com;
FileEndpoint=https://myaccount.file.core.windows.net;
QueueEndpoint=https://myaccount.queue.core.windows.net;
TableEndpoint=https://myaccount.table.core.windows.net;
AccountName=storagesample;
AccountKey=<account-key>
```

The endpoint values in a connection string are used to construct the request URIs to the storage services, and dictate the form of any URIs that are returned to your code.

If you've mapped a storage endpoint to a custom domain and omit that endpoint from a connection string, then you will not be able to use that connection string to access data in that service from your code.

For more information about configuring a custom domain for Azure Storage, see [Map a custom domain to an Azure Blob Storage endpoint](#).

## IMPORTANT

Service endpoint values in your connection strings must be well-formed URLs, including `https://` (recommended) or `http://`.

### Create a connection string with an endpoint suffix

To create a connection string for a storage service in regions or instances with different endpoint suffixes, such as for Azure China 21Vianet or Azure Government, use the following connection string format. Indicate whether you want to connect to the storage account through HTTPS (recommended) or HTTP, replace `myAccountName` with the name of your storage account, replace `myAccountKey` with your account access key, and replace `mySuffix` with the URI suffix:

```
DefaultEndpointsProtocol=[http|https];
AccountName=myAccountName;
AccountKey=myAccountKey;
EndpointSuffix=mySuffix;
```

Here's an example connection string for storage services in Azure China 21Vianet:

```
DefaultEndpointsProtocol=https;
AccountName=storagesample;
AccountKey=<account-key>;
EndpointSuffix=core.chinacloudapi.cn;
```

## Parsing a connection string

The [Microsoft Azure Configuration Manager Library for .NET](#) provides a class for parsing a connection string from a configuration file. The [CloudConfigurationManager](#) class parses configuration settings. It parses settings for client applications that run on the desktop, on a mobile device, in an Azure virtual machine, or in an Azure cloud service.

To reference the `CloudConfigurationManager` package, add the following `using` directives:

```
using Microsoft.Azure; //Namespace for CloudConfigurationManager
using Microsoft.Azure.Storage;
```

Here's an example that shows how to retrieve a connection string from a configuration file:

```
// Parse the connection string and return a reference to the storage account.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
 CloudConfigurationManager.GetSetting("StorageConnectionString"));
```

Using the Azure Configuration Manager is optional. You can also use an API such as the .NET Framework's  [ConfigurationManager Class](#).

## Next steps

- [Use the Azurite emulator for local Azure Storage development](#)
- [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#)

# Call REST API operations with Shared Key authorization

8/22/2022 • 16 minutes to read • [Edit Online](#)

This article shows you how to call the Azure Storage REST APIs, including how to form the Authorization header. It's written from the point of view of a developer who knows nothing about REST and no idea how to make a REST call. After you learn how to call a REST operation, you can leverage this knowledge to use any other Azure Storage REST operations.

## Prerequisites

The sample application lists the blob containers for a storage account. To try out the code in this article, you need the following items:

- Install [Visual Studio 2019](#) with the **Azure development** workload.
- An Azure subscription. If you don't have an Azure subscription, create a [free account](#) before you begin.
- A general-purpose storage account. If you don't yet have a storage account, see [Create a storage account](#).
- The example in this article shows how to list the containers in a storage account. To see output, add some containers to blob storage in the storage account before you start.

## Download the sample application

The sample application is a console application written in C#.

Use [git](#) to download a copy of the application to your development environment.

```
git clone https://github.com/Azure-Samples/storage-dotnet-rest-api-with-auth.git
```

This command clones the repository to your local git folder. To open the Visual Studio solution, look for the storage-dotnet-rest-api-with-auth folder, open it, and double-click on StorageRestApiAuth.sln.

## About REST

REST stands for *representational state transfer*. For a specific definition, check out [Wikipedia](#).

REST is an architecture that enables you to interact with a service over an internet protocol, such as HTTP/HTTPS. REST is independent of the software running on the server or the client. The REST API can be called from any platform that supports HTTP/HTTPS. You can write an application that runs on a Mac, Windows, Linux, an Android phone or tablet, iPhone, iPod, or web site, and use the same REST API for all of those platforms.

A call to the REST API consists of a request, which is made by the client, and a response, which is returned by the service. In the request, you send a URL with information about which operation you want to call, the resource to act upon, any query parameters and headers, and depending on the operation that was called, a payload of data. The response from the service includes a status code, a set of response headers, and depending on the operation that was called, a payload of data.

## About the sample application

The sample application lists the containers in a storage account. Once you understand how the information in the REST API documentation correlates to your actual code, other REST calls are easier to figure out.

If you look at the [Blob Service REST API](#), you see all of the operations you can perform on blob storage. The storage client libraries are wrappers around the REST APIs – they make it easy for you to access storage without using the REST APIs directly. But as noted above, sometimes you want to use the REST API instead of a storage client library.

## List Containers operation

Review the reference for the [ListContainers](#) operation. This information will help you understand where some of the fields come from in the request and response.

**Request Method:** GET. This verb is the HTTP method you specify as a property of the request object. Other values for this verb include HEAD, PUT, and DELETE, depending on the API you are calling.

**Request URI:** `https://myaccount.blob.core.windows.net/?comp=list`. The request URI is created from the blob storage account endpoint `https://myaccount.blob.core.windows.net` and the resource string `/?comp=list`.

**URI parameters:** There are additional query parameters you can use when calling ListContainers. A couple of these parameters are *timeout* for the call (in seconds) and *prefix*, which is used for filtering.

Another helpful parameter is *maxresults*: if more containers are available than this value, the response body will contain a *NextMarker* element that indicates the next container to return on the next request. To use this feature, you provide the *NextMarker* value as the *marker* parameter in the URI when you make the next request. When using this feature, it is analogous to paging through the results.

To use additional parameters, append them to the resource string with the value, like this example:

```
?comp=list&timeout=60&maxresults=100
```

**Request Headers:** This section lists the required and optional request headers. Three of the headers are required: an *Authorization* header, *x-ms-date* (contains the UTC time for the request), and *x-ms-version* (specifies the version of the REST API to use). Including *x-ms-client-request-id* in the headers is optional – you can set the value for this field to anything; it is written to the storage analytics logs when logging is enabled.

**Request Body:** There is no request body for ListContainers. Request Body is used on all of the PUT operations when uploading blobs, as well as SetContainerAccessPolicy, which allows you to send in an XML list of stored access policies to apply. Stored access policies are discussed in the article [Using Shared Access Signatures \(SAS\)](#).

**Response Status Code:** Tells of any status codes you need to know. In this example, an HTTP status code of 200 is ok. For a complete list of HTTP status codes, check out [Status Code Definitions](#). To see error codes specific to the Storage REST APIs, see [Common REST API error codes](#)

**Response Headers:** These include *Content Type*, *x-ms-request-id*, which is the request ID you passed in; *x-ms-version*, which indicates the version of the Blob service used; and the *Date*, which is in UTC and tells what time the request was made.

**Response Body:** This field is an XML structure providing the data requested. In this example, the response is a list of containers and their properties.

## Creating the REST request

For security when running in production, always use HTTPS rather than HTTP. For the purposes of this exercise, you should use HTTP so you can view the request and response data. To view the request and response information in the actual REST calls, you can download [Fiddler](#) or a similar application. In the Visual Studio

solution, the storage account name and key are hardcoded in the class. The `ListContainersAsyncREST` method passes the storage account name and storage account key to the methods that are used to create the various components of the REST request. In a real world application, the storage account name and key would reside in a configuration file, environment variables, or be retrieved from an Azure Key Vault.

In our sample project, the code for creating the Authorization header is in a separate class. The idea is that you could take the whole class and add it to your own solution and use it "as is." The Authorization header code works for most REST API calls to Azure Storage.

To build the request, which is an `HttpRequestMessage` object, go to `ListContainersAsyncREST` in `Program.cs`. The steps for building the request are:

- Create the URI to be used for calling the service.
- Create the `HttpRequestMessage` object and set the payload. The payload is null for `ListContainersAsyncREST` because we're not passing anything in.
- Add the request headers for `x-ms-date` and `x-ms-version`.
- Get the authorization header and add it.

Some basic information you need:

- For `ListContainers`, the `method` is `GET`. This value is set when instantiating the request.
- The `resource` is the query portion of the URI that indicates which API is being called, so the value is `/?comp=list`. As noted earlier, the resource is on the reference documentation page that shows the information about the [ListContainers API](#).
- The URI is constructed by creating the Blob service endpoint for that storage account and concatenating the resource. The value for `request URI` ends up being `http://contosorest.blob.core.windows.net/?comp=list`.
- For `ListContainers`, `requestBody` is null and there are no extra `headers`.

Different APIs may have other parameters to pass in such as `ifMatch`. An example of where you might use `ifMatch` is when calling `PutBlob`. In that case, you set `ifMatch` to an eTag, and it only updates the blob if the eTag you provide matches the current eTag on the blob. If someone else has updated the blob since retrieving the eTag, their change won't be overridden.

First, set the `uri` and the `payload`.

```
// Construct the URI. It will look like this:
// https://myaccount.blob.core.windows.net/resource
String uri = string.Format("http://{0}.blob.core.windows.net?comp=list", storageAccountName);

// Provide the appropriate payload, in this case null.
// we're not passing anything in.
Byte[] requestPayload = null;
```

Next, instantiate the request, setting the method to `GET` and providing the URI.

```
// Instantiate the request message with a null payload.
using (var httpRequestMessage = new HttpRequestMessage(HttpMethod.Get, uri)
{ Content = (requestPayload == null) ? null : new ByteArrayContent(requestPayload) })
{
```

Add the request headers for `x-ms-date` and `x-ms-version`. This place in the code is also where you add any additional request headers required for the call. In this example, there are no additional headers. An example of an API that passes in extra headers is the Set Container ACL operation. This API call adds a header called "x-ms-blob-public-access" and the value for the access level.

```
// Add the request headers for x-ms-date and x-ms-version.
DateTime now = DateTime.UtcNow;
httpRequestMessage.Headers.Add("x-ms-date", now.ToString("R", CultureInfo.InvariantCulture));
httpRequestMessage.Headers.Add("x-ms-version", "2017-07-29");
// If you need any additional headers, add them here before creating
// the authorization header.
```

Call the method that creates the authorization header and add it to the request headers. You'll see how to create the authorization header later in the article. The method name is GetAuthorizationHeader, which you can see in this code snippet:

```
// Get the authorization header and add it.
httpRequestMessage.Headers.Authorization = AzureStorageAuthenticationHelper.GetAuthorizationHeader(
 storageAccountName, storageAccountKey, now, httpRequestMessage);
```

At this point, `httpRequestMessage` contains the REST request complete with the authorization headers.

## Send the request

Now that you have constructed the request, you can call the `SendAsync` method to send it to Azure Storage. Check that the value of the response status code is 200, meaning that the operation has succeeded. Next, parse the response. In this case, you get an XML list of containers. Let's look at the code for calling the `GetRESTRequest` method to create the request, execute the request, and then examine the response for the list of containers.

```
// Send the request.
using (HttpResponseMessage httpResponseMessage =
 await new HttpClient().SendAsync(httpRequestMessage, cancellationToken))
{
 // If successful (status code = 200),
 // parse the XML response for the container names.
 if (httpResponseMessage.StatusCode == HttpStatusCode.OK)
 {
 String xmlString = await httpResponseMessage.Content.ReadAsStringAsync();
 XElement x = XElement.Parse(xmlString);
 foreach (XElement container in x.Element("Containers").Elements("Container"))
 {
 Console.WriteLine("Container name = {0}", container.Element("Name").Value);
 }
 }
}
```

If you run a network sniffer such as [Fiddler](#) when making the call to `SendAsync`, you can see the request and response information. Let's take a look. The name of the storage account is *contosorest*.

### Request:

```
GET /?comp=list HTTP/1.1
```

### Request Headers:

```
x-ms-date: Thu, 16 Nov 2017 23:34:04 GMT
x-ms-version: 2014-02-14
Authorization: SharedKey contosorest:1dV1YJWWJAOSHTCPGiwdX1rOS8B4fenYP/VrU0LfzQk=
Host: contosorest.blob.core.windows.net
Connection: Keep-Alive
```

## Status code and response headers returned after execution:

```
HTTP/1.1 200 OK
Content-Type: application/xml
Server: Windows-Azure-Blob/1.0 Microsoft-HTTPAPI/2.0
x-ms-request-id: 3e889876-001e-0039-6a3a-5f4396000000
x-ms-version: 2017-07-29
Date: Fri, 17 Nov 2017 00:23:42 GMT
Content-Length: 1511
```

**Response body (XML):** For the List Containers operation, this shows the list of containers and their properties.

```
<?xml version="1.0" encoding="utf-8"?>
<EnumerationResults
 ServiceEndpoint="http://contosorest.blob.core.windows.net/">
 <Containers>
 <Container>
 <Name>container-1</Name>
 <Properties>
 <Last-Modified>Thu, 16 Mar 2017 22:39:48 GMT</Last-Modified>
 <Etag>"0x8D46CBD5A7C301D"</Etag>
 <LeaseStatus>unlocked</LeaseStatus>
 <LeaseState>available</LeaseState>
 </Properties>
 </Container>
 <Container>
 <Name>container-2</Name>
 <Properties>
 <Last-Modified>Thu, 16 Mar 2017 22:40:50 GMT</Last-Modified>
 <Etag>"0x8D46CBD7F49E9BD"</Etag>
 <LeaseStatus>unlocked</LeaseStatus>
 <LeaseState>available</LeaseState>
 </Properties>
 </Container>
 <Container>
 <Name>container-3</Name>
 <Properties>
 <Last-Modified>Thu, 16 Mar 2017 22:41:10 GMT</Last-Modified>
 <Etag>"0x8D46CBD8B243D68"</Etag>
 <LeaseStatus>unlocked</LeaseStatus>
 <LeaseState>available</LeaseState>
 </Properties>
 </Container>
 <Container>
 <Name>container-4</Name>
 <Properties>
 <Last-Modified>Thu, 16 Mar 2017 22:41:25 GMT</Last-Modified>
 <Etag>"0x8D46CBD93FED46F"</Etag>
 <LeaseStatus>unlocked</LeaseStatus>
 <LeaseState>available</LeaseState>
 </Properties>
 </Container>
 <Container>
 <Name>container-5</Name>
 <Properties>
 <Last-Modified>Thu, 16 Mar 2017 22:41:39 GMT</Last-Modified>
 <Etag>"0x8D46CBD9C762815"</Etag>
 <LeaseStatus>unlocked</LeaseStatus>
 <LeaseState>available</LeaseState>
 </Properties>
 </Container>
 </Containers>
 <NextMarker />
</EnumerationResults>
```

Now that you understand how to create the request, call the service, and parse the results, let's see how to create the authorization header. Creating that header is complicated, but the good news is that once you have the code working, it works for all of the Storage Service REST APIs.

## Creating the authorization header

### TIP

Azure Storage now supports Azure Active Directory (Azure AD) integration for blobs and queues. Azure AD offers a much simpler experience for authorizing a request to Azure Storage. For more information on using Azure AD to authorize REST operations, see [Authorize with Azure Active Directory](#). For an overview of Azure AD integration with Azure Storage, see [Authenticate access to Azure Storage using Azure Active Directory](#).

There is an article that explains conceptually (no code) how to [Authorize requests to Azure Storage](#).

Let's distill that article down to exactly what is needed and show the code.

First, use Shared Key authorization. The authorization header format looks like this:

```
Authorization="SharedKey <storage account name>:<signature>"
```

The signature field is a Hash-based Message Authentication Code (HMAC) created from the request and calculated using the SHA256 algorithm, then encoded using Base64 encoding. Got that? (Hang in there, you haven't even heard the word *canonicalized* yet.)

This code snippet shows the format of the Shared Key signature string:

```
StringToSign = VERB + "\n" +
 Content-Encoding + "\n" +
 Content-Language + "\n" +
 Content-Length + "\n" +
 Content-MD5 + "\n" +
 Content-Type + "\n" +
 Date + "\n" +
 If-Modified-Since + "\n" +
 If-Match + "\n" +
 If-None-Match + "\n" +
 If-Unmodified-Since + "\n" +
 Range + "\n" +
 CanonicalizedHeaders +
 CanonicalizedResource;
```

Most of these fields are rarely used. For Blob storage, you specify VERB, md5, content length, Canonicalized Headers, and Canonicalized Resource. You can leave the others blank (but put in the `\n` so it knows they are blank).

What are CanonicalizedHeaders and CanonicalizedResource? Good question. In fact, what does canonicalized mean? Microsoft Word doesn't even recognize it as a word. Here's what [Wikipedia says about canonicalization](#): *In computer science, canonicalization (sometimes standardization or normalization) is a process for converting data that has more than one possible representation into a "standard", "normal" or canonical form.* In normal-speak, this means to take the list of items (such as headers in the case of Canonicalized Headers) and standardize them into a required format. Basically, Microsoft decided on a format and you need to match it.

Let's start with those two canonicalized fields, because they are required to create the Authorization header.

### Canonicalized headers

To create this value, retrieve the headers that start with "x-ms-" and sort them, then format them into a string of [key:value\n] instances, concatenated into one string. For this example, the canonicalized headers look like this:

```
x-ms-date:Fri, 17 Nov 2017 00:44:48 GMT\nx-ms-version:2017-07-29\n
```

Here's the code used to create that output:

```
private static string GetCanonicalizedHeaders(HttpRequestMessage httpRequestMessage)
{
 var headers = from kvp in httpRequestMessage.Headers
 where kvp.Key.StartsWith("x-ms-", StringComparison.OrdinalIgnoreCase)
 orderby kvp.Key
 select new { Key = kvp.Key.ToLowerInvariant(), kvp.Value };

 StringBuilder headersBuilder = new StringBuilder();

 // Create the string in the right format; this is what makes the headers "canonicalized" --
 // it means put in a standard format. https://en.wikipedia.org/wiki/Canonicalization
 foreach (var kvp in headers)
 {
 headersBuilder.Append(kvp.Key);
 char separator = ':';

 // Get the value for each header, strip out \r\n if found, then append it with the key.
 foreach (string headerValue in kvp.Value)
 {
 string trimmedValue = headerValue.TrimStart().Replace("\r\n", string.Empty);
 headersBuilder.Append(separator).Append(trimmedValue);

 // Set this to a comma; this will only be used
 // if there are multiple values for one of the headers.
 separator = ',';
 }

 headersBuilder.Append("\n");
 }

 return headersBuilder.ToString();
}
```

## Canonicalized resource

This part of the signature string represents the storage account targeted by the request. Remember that the Request URI is http://contosorest.blob.core.windows.net/?comp=list, with the actual account name (contosorest in this case). In this example, this is returned:

```
/contosorest/\ncomp:list
```

If you have query parameters, this example includes those parameters as well. Here's the code, which also handles additional query parameters and query parameters with multiple values. Remember that you're building this code to work for all of the REST APIs. You want to include all possibilities, even if the ListContainers method doesn't need all of them.

```
private static string GetCanonicalizedResource(Uri address, string storageAccountName)
{
 // The absolute path will be "/" because we're getting a list of containers.
 StringBuilder sb = new StringBuilder("/").Append(storageAccountName).Append(address.AbsolutePath);

 // Address.Query is the resource, such as "?comp=list".
 // This ends up with a NameValueCollection with 1 entry having key=comp, value=list.
 // It will have more entries if you have more query parameters.
 NameValueCollection values = HttpUtility.ParseQueryString(address.Query);

 foreach (var item in values.AllKeys.OrderBy(k => k))
 {
 sb.Append('\n').Append(item.ToLower()).Append(':').Append(values[item]);
 }

 return sb.ToString();
}
```

Now that the canonicalized strings are set, let's look at how to create the authorization header itself. You start by creating a string of the message signature in the format of StringToSign previously displayed in this article. This concept is easier to explain using comments in the code, so here it is, the final method that returns the Authorization Header:

```
internal static AuthenticationHeaderValue GetAuthorizationHeader(
 string storageAccountName, string storageAccountKey, DateTime now,
 HttpRequestMessage httpRequestMessage, string ifMatch = "", string md5 = "")
{
 // This is the raw representation of the message signature.
 HttpMethod method = httpRequestMessage.Method;
 String MessageSignature = String.Format("{0}\n\n\n{1}\n{5}\n\n\n{2}\n\n\n{3}{4}",
 method.ToString(),
 (method == HttpMethod.Get || method == HttpMethod.Head) ? String.Empty
 : httpRequestMessage.Content.Headers.ContentLength.ToString(),
 ifMatch,
 GetCanonicalizedHeaders(httpRequestMessage),
 GetCanonicalizedResource(httpRequestMessage.RequestUri, storageAccountName),
 md5);

 // Now turn it into a byte array.
 byte[] SignatureBytes = Encoding.UTF8.GetBytes(MessageSignature);

 // Create the HMACSHA256 version of the storage key.
 HMACSHA256 SHA256 = new HMACSHA256(Convert.FromBase64String(storageAccountKey));

 // Compute the hash of the SignatureBytes and convert it to a base64 string.
 string signature = Convert.ToBase64String(SHA256.ComputeHash(SignatureBytes));

 // This is the actual header that will be added to the list of request headers.
 AuthenticationHeaderValue authHV = new AuthenticationHeaderValue("SharedKey",
 storageAccountName + ":" + signature);
 return authHV;
}
```

When you run this code, the resulting MessageSignature looks like this example:

```
GET\n\n\n\n\n\n\n\n\n\n\n\n\n\nnx-ms-date:Fri, 17 Nov 2017 01:07:37 GMT\nnx-ms-version:2017-07-29\n/n/contosorest/\nncmp:list
```

Here's the final value for AuthorizationHeader:

```
SharedKey contosorest:Ms5sfwkA8nqTRw7Uury4MPHqM6Rj2nfgbYNvUK0a67w=
```

The AuthorizationHeader is the last header placed in the request headers before posting the response.

That covers everything you need to know to put together a class with which you can create a request to call the Storage Services REST APIs.

## Example: List blobs

Let's look at how to change the code to call the List Blobs operation for container *container-1*. This code is almost identical to the code for listing containers, the only differences being the URI and how you parse the response.

If you look at the reference documentation for [ListBlobs](#), you find that the method is *GET* and the RequestURI is:

```
https://myaccount.blob.core.windows.net/container-1?restype=container&comp=list
```

In ListContainersAsyncREST, change the code that sets the URI to the API for ListBlobs. The container name is **container-1**.

```
String uri =
 string.Format("http://{0}.blob.core.windows.net/container-1?restype=container&comp=list",
 storageAccountName);
```

Then where you handle the response, change the code to look for blobs instead of containers.

```
foreach (XElement container in x.Element("Blobs").Elements("Blob"))
{
 Console.WriteLine("Blob name = {0}", container.Element("Name").Value);
}
```

When you run this sample, you get results like the following:

### Canonicalized headers:

```
x-ms-date:Fri, 17 Nov 2017 05:16:48 GMT\nx-ms-version:2017-07-29\n
```

### Canonicalized resource:

```
/contosorest/container-1\ncomp:list\nrestype:container
```

### Message signature:

```
GET\n\n\n\n\n\n\n\n\n\nnx-ms-date:Fri, 17 Nov 2017 05:16:48 GMT
\nx-ms-version:2017-07-29\n/contosorest/container-1\ncomp:list\nrestype:container
```

### Authorization header:

```
SharedKey contosorest:uvwxyz1WUIv2LYC6e3En10/7EIQJ5X9KtFQqrZkxi6s=
```

The following values are from [Fiddler](#):

**Request:**

```
GET http://contosorest.blob.core.windows.net/container-1?restype=container&comp=list HTTP/1.1
```

**Request Headers:**

```
x-ms-date: Fri, 17 Nov 2017 05:16:48 GMT
x-ms-version: 2017-07-29
Authorization: SharedKey contosorest:uvwxyz1WUIv2LYC6e3En10/7EIQJ5X9KtFQqrZkxi6s=
Host: contosorest.blob.core.windows.net
Connection: Keep-Alive
```

**Status code and response headers returned after execution:**

```
HTTP/1.1 200 OK
Content-Type: application/xml
Server: Windows-Azure-Blob/1.0 Microsoft-HTTPAPI/2.0
x-ms-request-id: 7e9316da-001e-0037-4063-5faf9d000000
x-ms-version: 2017-07-29
Date: Fri, 17 Nov 2017 05:20:21 GMT
Content-Length: 1135
```

**Response body (XML):** This XML response shows the list of blobs and their properties.

```

<?xml version="1.0" encoding="utf-8"?>
<EnumerationResults
 ServiceEndpoint="http://contosorest.blob.core.windows.net/" ContainerName="container-1">
 <Blobs>
 <Blob>
 <Name>DogInCatTree.png</Name>
 <Properties><Last-Modified>Fri, 17 Nov 2017 01:41:14 GMT</Last-Modified>
 <Etag>0x8D52D5C4A4C96B0</Etag>
 <Content-Length>419416</Content-Length>
 <Content-Type>image/png</Content-Type>
 <Content-Encoding />
 <Content-Language />
 <Content-MD5 />
 <Cache-Control />
 <Content-Disposition />
 <BlobType>BlockBlob</BlobType>
 <LeaseStatus>unlocked</LeaseStatus>
 <LeaseState>available</LeaseState>
 <ServerEncrypted>true</ServerEncrypted>
 </Properties>
 </Blob>
 <Blob>
 <Name>GuyEyeingOreos.png</Name>
 <Properties>
 <Last-Modified>Fri, 17 Nov 2017 01:41:14 GMT</Last-Modified>
 <Etag>0x8D52D5C4A25A6F6</Etag>
 <Content-Length>167464</Content-Length>
 <Content-Type>image/png</Content-Type>
 <Content-Encoding />
 <Content-Language />
 <Content-MD5 />
 <Cache-Control />
 <Content-Disposition />
 <BlobType>BlockBlob</BlobType>
 <LeaseStatus>unlocked</LeaseStatus>
 <LeaseState>available</LeaseState>
 <ServerEncrypted>true</ServerEncrypted>
 </Properties>
 </Blob>
 </Blobs>
 <NextMarker />
</EnumerationResults>

```

## Summary

In this article, you learned how to make a request to the blob storage REST API. With the request, you can retrieve a list of containers or a list of blobs in a container. You learned how to create the authorization signature for the REST API call and how to use it in the REST request. Finally, you learned how to examine the response.

## Next steps

- [Blob Service REST API](#)
- [File Service REST API](#)
- [Queue Service REST API](#)
- [Table Service REST API](#)

# Prevent Shared Key authorization for an Azure Storage account

8/22/2022 • 15 minutes to read • [Edit Online](#)

Every secure request to an Azure Storage account must be authorized. By default, requests can be authorized with either Azure Active Directory (Azure AD) credentials, or by using the account access key for Shared Key authorization. Of these two types of authorization, Azure AD provides superior security and ease of use over Shared Key, and is recommended by Microsoft. To require clients to use Azure AD to authorize requests, you can disallow requests to the storage account that are authorized with Shared Key.

When you disallow Shared Key authorization for a storage account, Azure Storage rejects all subsequent requests to that account that are authorized with the account access keys. Only secured requests that are authorized with Azure AD will succeed. For more information about using Azure AD, see [Authorize access to data in Azure Storage](#).

This article describes how to detect requests sent with Shared Key authorization and how to remediate Shared Key authorization for your storage account.

## Detect the type of authorization used by client applications

When you disallow Shared Key authorization for a storage account, requests from clients that are using the account access keys for Shared Key authorization will fail. To understand how disallowing Shared Key authorization may affect client applications before you make this change, enable logging and metrics for the storage account. You can then analyze patterns of requests to your account over a period of time to determine how requests are being authorized.

Use metrics to determine how many requests the storage account is receiving that are authorized with Shared Key or a shared access signature (SAS). Use logs to determine which clients are sending those requests.

A SAS may be authorized with either Shared Key or Azure AD. For more information about interpreting requests made with a shared access signature, see [Understand how disallowing Shared Key affects SAS tokens](#).

### Monitor how many requests are authorized with Shared Key

To track how requests to a storage account are being authorized, use Azure Metrics Explorer in the Azure portal. For more information about Metrics Explorer, see [Getting started with Azure Metrics Explorer](#).

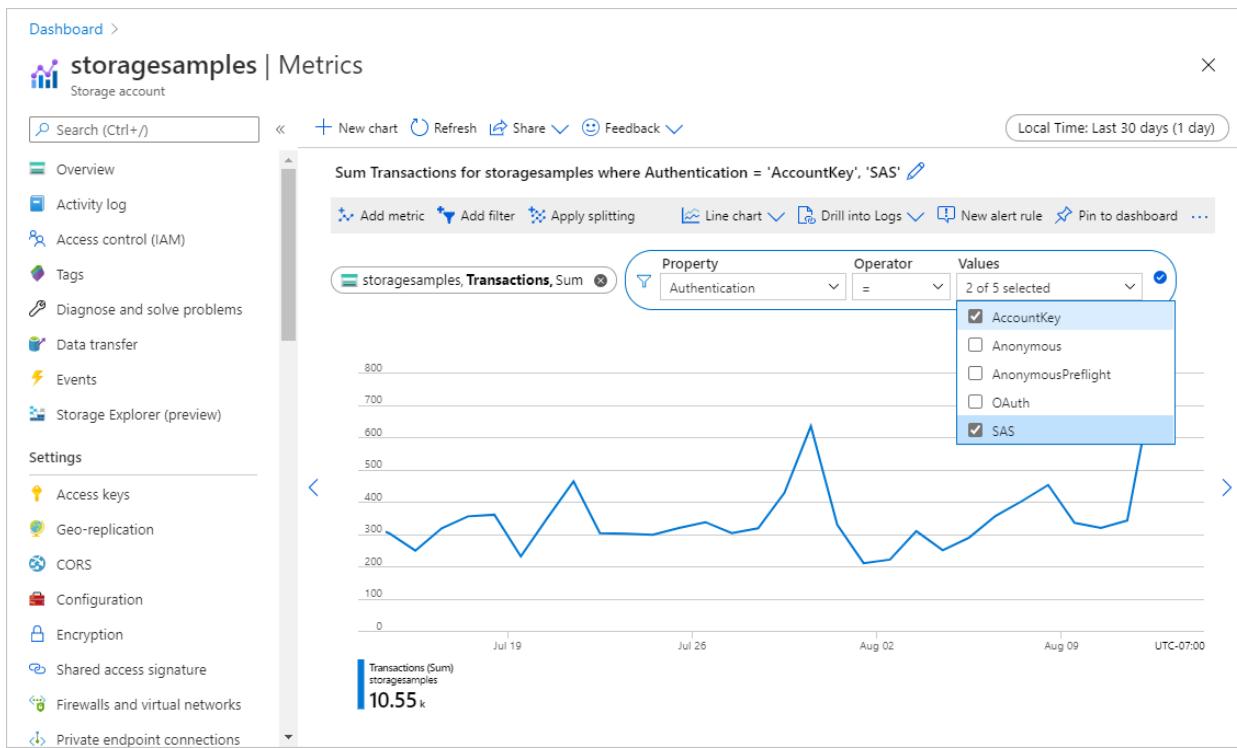
Follow these steps to create a metric that tracks requests made with Shared Key or SAS:

1. Navigate to your storage account in the Azure portal. Under the **Monitoring** section, select **Metrics**.
2. Select **Add metric**. In the **Metric** dialog, specify the following values:
  - a. Leave the **Scope** field set to the name of the storage account.
  - b. Set the **Metric Namespace** to *Account*. This metric will report on all requests against the storage account.
  - c. Set the **Metric** field to *Transactions*.
  - d. Set the **Aggregation** field to *Sum*.

The new metric will display the sum of the number of transactions against the storage account over a given interval of time. The resulting metric appears as shown in the following image:

3. Next, select the **Add filter** button to create a filter on the metric for type of authorization.
4. In the **Filter** dialog, specify the following values:
  - a. Set the **Property** value to *Authentication*.
  - b. Set the **Operator** field to the equal sign (=).
  - c. In the **Values** field, select *Account Key* and *SAS*.
5. In the upper-right corner, select the time range for which you want to view the metric. You can also indicate how granular the aggregation of requests should be, by specifying intervals anywhere from 1 minute to 1 month. For example, set the **Time range** to 30 days and the **Time granularity** to 1 day to see requests aggregated by day over the past 30 days.

After you have configured the metric, requests to your storage account will begin to appear on the graph. The following image shows requests that were authorized with Shared Key or made with a SAS token. Requests are aggregated per day over the past thirty days.



You can also configure an alert rule to notify you when a certain number of requests that are authorized with Shared Key are made against your storage account. For more information, see [Create, view, and manage metric alerts using Azure Monitor](#).

### Analyze logs to identify clients that are authorizing requests with Shared Key or SAS

Azure Storage logs capture details about requests made against the storage account, including how a request was authorized. You can analyze the logs to determine which clients are authorizing requests with Shared Key or a SAS token.

To log requests to your Azure Storage account in order to evaluate how they are authorized, you can use Azure Storage logging in Azure Monitor (preview). For more information, see [Monitor Azure Storage](#).

Azure Storage logging in Azure Monitor supports using log queries to analyze log data. To query logs, you can use an Azure Log Analytics workspace. To learn more about log queries, see [Tutorial: Get started with Log Analytics queries](#).

## Create a diagnostic setting in the Azure portal

To log Azure Storage data with Azure Monitor and analyze it with Azure Log Analytics, you must first create a diagnostic setting that indicates what types of requests and for which storage services you want to log data. To create a diagnostic setting in the Azure portal, follow these steps:

1. Create a new Log Analytics workspace in the subscription that contains your Azure Storage account, or use an existing Log Analytics workspace. After you configure logging for your storage account, the logs will be available in the Log Analytics workspace. For more information, see [Create a Log Analytics workspace in the Azure portal](#).
2. Navigate to your storage account in the Azure portal.
3. In the Monitoring section, select **Diagnostic settings (preview)**.
4. Select the Azure Storage service for which you want to log requests. For example, choose **Blob** to log requests to Blob storage.
5. Select **Add diagnostic setting**.
6. Provide a name for the diagnostic setting.
7. Under **Category details**, in the **log** section, choose **StorageRead**, **StorageWrite**, and **StorageDelete** to log all data requests to the selected service.
8. Under **Destination details**, select **Send to Log Analytics**. Select your subscription and the Log Analytics workspace you created earlier, as shown in the following image.

The screenshot shows the 'Diagnostics setting' blade in the Azure portal. At the top, there are buttons for Save, Discard, Delete, and Provide feedback. Below that, a description explains what a diagnostic setting is: 'A diagnostic setting specifies a list of categories of platform logs and/or metrics that you want to collect from a resource, and one or more destinations that you would stream them to. Normal usage charges for the destination will occur.' A link to 'Learn more about the different log categories and contents of those logs' is provided. The main area is divided into 'Category details' and 'Destination details'. In 'Category details', under 'log', three checkboxes are checked: 'StorageRead', 'StorageWrite', and 'StorageDelete'. Under 'metric', one checkbox is unchecked: 'Transaction'. In 'Destination details', the 'Send to Log Analytics' checkbox is checked. The 'Subscription' dropdown is set to 'Azure Storage content development and testing'. The 'Log Analytics workspace' dropdown is set to 'AuthLogsWorkspace (westus)'. There are also two unchecked checkboxes: 'Archive to a storage account' and 'Stream to an event hub'.

You can create a diagnostic setting for each type of Azure Storage resource in your storage account.

After you create the diagnostic setting, requests to the storage account are subsequently logged according to that setting. For more information, see [Create diagnostic setting to collect resource logs and metrics in Azure](#).

For a reference of fields available in Azure Storage logs in Azure Monitor, see [Resource logs \(preview\)](#).

### Query logs for requests made with Shared Key or SAS

Azure Storage logs in Azure Monitor include the type of authorization that was used to make a request to a storage account. To retrieve logs for requests made in the last seven days that were authorized with Shared Key or SAS, open your Log Analytics workspace. Next, paste the following query into a new log query and run it. This query displays the ten IP addresses that most frequently sent requests that were authorized with Shared Key or SAS:

```
StorageBlobLogs
| where AuthenticationType in ("AccountKey", "SAS") and TimeGenerated > ago(7d)
| summarize count() by CallerIpAddress, UserAgentHeader, AccountName
| top 10 by count_ desc
```

You can also configure an alert rule based on this query to notify you about requests authorized with Shared Key or SAS. For more information, see [Create, view, and manage log alerts using Azure Monitor](#).

## Remediate authorization via Shared Key

After you have analyzed how requests to your storage account are being authorized, you can take action to prevent access via Shared Key. But first, you need to update any applications that are using Shared Key authorization to use Azure AD instead. You can monitor logs and metrics as described in [Detect the type of authorization used by client applications](#) to track the transition. For more information about using Azure AD to access data in a storage account, see [Authorize access to data in Azure Storage](#).

When you are confident that you can safely reject requests that are authorized with Shared Key, you can set the **AllowSharedKeyAccess** property for the storage account to **false**.

The **AllowSharedKeyAccess** property is not set by default and does not return a value until you explicitly set it. The storage account permits requests that are authorized with Shared Key when the property value is **null** or when it is **true**.

### WARNING

If any clients are currently accessing data in your storage account with Shared Key, then Microsoft recommends that you migrate those clients to Azure AD before disallowing Shared Key access to the storage account.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To disallow Shared Key authorization for a storage account in the Azure portal, follow these steps:

1. Navigate to your storage account in the Azure portal.
2. Locate the **Configuration** setting under **Settings**.
3. Set **Allow storage account key access** to **Disabled**.

Allow storage account key access ⓘ  
 Disabled  Enabled

**⚠** When Allow storage account key access is disabled, any requests to the account that are authorized with Shared Key, including shared access signatures (SAS), will be denied. Client applications that currently access the storage account using Shared Key will no longer work. [Learn more about Allow storage account key access ↗](#)

After you disallow Shared Key authorization, making a request to the storage account with Shared Key authorization will fail with error code 403 (Forbidden). Azure Storage returns error indicating that key-based authorization is not permitted on the storage account.

The **AllowSharedKeyAccess** property is supported for storage accounts that use the Azure Resource Manager deployment model only. For information about which storage accounts use the Azure Resource Manager deployment model, see [Types of storage accounts](#).

### Verify that Shared Key access is not allowed

To verify that Shared Key authorization is no longer permitted, you can attempt to call a data operation with the account access key. The following example attempts to create a container using the access key. This call will fail when Shared Key authorization is disallowed for the storage account. Remember to replace the placeholder values in brackets with your own values:

```
az storage container create \
--account-name <storage-account> \
--name sample-container \
--account-key <key> \
--auth-mode key
```

#### NOTE

Anonymous requests are not authorized and will proceed if you have configured the storage account and container for anonymous public read access. For more information, see [Configure anonymous public read access for containers and blobs](#).

### Check the Shared Key access setting for multiple accounts

To check the Shared Key access setting across a set of storage accounts with optimal performance, you can use the Azure Resource Graph Explorer in the Azure portal. To learn more about using the Resource Graph Explorer, see [Quickstart: Run your first Resource Graph query using Azure Resource Graph Explorer](#).

Running the following query in the Resource Graph Explorer returns a list of storage accounts and displays the Shared Key access setting for each account:

```
resources
| where type == 'Microsoft.Storage/storageAccounts'
| extend allowSharedKeyAccess = parse_json(properties).allowSharedKeyAccess
| project subscriptionId, resourceGroup, name, allowSharedKeyAccess
```

## Permissions for allowing or disallowing Shared Key access

To set the **AllowSharedKeyAccess** property for the storage account, a user must have permissions to create and manage storage accounts. Azure role-based access control (Azure RBAC) roles that provide these permissions include the **Microsoft.Storage/storageAccounts/write** or **Microsoft.Storage/storageAccounts/\*** action. Built-in roles with this action include:

- The Azure Resource Manager [Owner](#) role
- The Azure Resource Manager [Contributor](#) role
- The [Storage Account Contributor](#) role

These roles do not provide access to data in a storage account via Azure Active Directory (Azure AD). However, they include the **Microsoft.Storage/storageAccounts/listkeys/action**, which grants access to the account access keys. With this permission, a user can use the account access keys to access all data in a storage account.

Role assignments must be scoped to the level of the storage account or higher to permit a user to allow or disallow Shared Key access for the storage account. For more information about role scope, see [Understand scope for Azure RBAC](#).

Be careful to restrict assignment of these roles only to those who require the ability to create a storage account or update its properties. Use the principle of least privilege to ensure that users have the fewest permissions that they need to accomplish their tasks. For more information about managing access with Azure RBAC, see [Best practices for Azure RBAC](#).

#### NOTE

The classic subscription administrator roles Service Administrator and Co-Administrator include the equivalent of the Azure Resource Manager [Owner](#) role. The [Owner](#) role includes all actions, so a user with one of these administrative roles can also create and manage storage accounts. For more information, see [Classic subscription administrator roles, Azure roles, and Azure AD administrator roles](#).

## Understand how disallowing Shared Key affects SAS tokens

When Shared Key access is disallowed for the storage account, Azure Storage handles SAS tokens based on the type of SAS and the service that is targeted by the request. The following table shows how each type of SAS is authorized and how Azure Storage will handle that SAS when the [AllowSharedKeyAccess](#) property for the storage account is **false**.

TYPE OF SAS	TYPE OF AUTHORIZATION	BEHAVIOR WHEN ALLOWSHAREDKYACCESS IS FALSE
User delegation SAS (Blob storage only)	Azure AD	Request is permitted. Microsoft recommends using a user delegation SAS when possible for superior security.
Service SAS	Shared Key	Request is denied for all Azure Storage services.
Account SAS	Shared Key	Request is denied for all Azure Storage services.

Azure metrics and logging in Azure Monitor do not distinguish between different types of shared access signatures. The [SAS](#) filter in Azure Metrics Explorer and the [SAS](#) field in Azure Storage logging in Azure Monitor both report requests that are authorized with any type of SAS. However, different types of shared access signatures are authorized differently, and behave differently when Shared Key access is disallowed:

- A service SAS token or an account SAS token is authorized with Shared Key and will not be permitted on a request to Blob storage when the [AllowSharedKeyAccess](#) property is set to **false**.
- A user delegation SAS is authorized with Azure AD and will be permitted on a request to Blob storage when the [AllowSharedKeyAccess](#) property is set to **false**.

When you are evaluating traffic to your storage account, keep in mind that metrics and logs as described in [Detect the type of authorization used by client applications](#) may include requests made with a user delegation SAS.

For more information about shared access signatures, see [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#).

## Consider compatibility with other Azure tools and services

A number of Azure services use Shared Key authorization to communicate with Azure Storage. If you disallow Shared Key authorization for a storage account, these services will not be able to access data in that account, and your applications may be adversely affected.

Some Azure tools offer the option to use Azure AD authorization to access Azure Storage. The following table lists some popular Azure tools and notes whether they can use Azure AD to authorize requests to Azure Storage.

AZURE TOOL	AZURE AD AUTHORIZATION TO AZURE STORAGE
Azure portal	Supported. For information about authorizing with your Azure AD account from the Azure portal, see <a href="#">Choose how to authorize access to blob data in the Azure portal</a> .
AzCopy	Supported for Blob storage. For information about authorizing AzCopy operations, see <a href="#">Choose how you'll provide authorization credentials</a> in the AzCopy documentation.
Azure Storage Explorer	Supported for Blob storage, Queue storage, Table storage and Azure Data Lake Storage Gen2. Azure AD access to File storage is not supported. Make sure to select the correct Azure AD tenant. For more information, see <a href="#">Get started with Storage Explorer</a>
Azure PowerShell	Supported. For information about how to authorize PowerShell commands for blob or queue operations with Azure AD, see <a href="#">Run PowerShell commands with Azure AD credentials to access blob data</a> or <a href="#">Run PowerShell commands with Azure AD credentials to access queue data</a> .
Azure CLI	Supported. For information about how to authorize Azure CLI commands with Azure AD for access to blob and queue data, see <a href="#">Run Azure CLI commands with Azure AD credentials to access blob or queue data</a> .
Azure IoT Hub	Supported. For more information, see <a href="#">IoT Hub support for virtual networks</a> .
Azure Cloud Shell	Azure Cloud Shell is an integrated shell in the Azure portal. Azure Cloud Shell hosts files for persistence in an Azure file share in a storage account. These files will become inaccessible if Shared Key authorization is disallowed for that storage account. For more information, see <a href="#">Connect your Microsoft Azure Files storage</a> .  To run commands in Azure Cloud Shell to manage storage accounts for which Shared Key access is disallowed, first make sure that you have been granted the necessary permissions to these accounts via Azure RBAC. For more information, see <a href="#">What is Azure role-based access control (Azure RBAC)?</a> .

## Disallow Shared Key authorization to use Azure AD Conditional Access

To protect an Azure Storage account with Azure AD [Conditional Access](#) policies, you must disallow Shared Key authorization for the storage account. Follow the steps described in [Detect the type of authorization used by client applications](#) to analyze the potential impact of this change for existing storage accounts before disallowing Shared Key authorization.

## Transition Azure Files and Table storage workloads

Azure Storage supports Azure AD authorization for requests to Blob and Queue storage only. If you disallow authorization with Shared Key for a storage account, requests to Azure Files or Table storage that use Shared

Key authorization will fail. Because the Azure portal always uses Shared Key authorization to access file and table data, if you disallow authorization with Shared Key for the storage account, you will not be able to access file or table data in the Azure portal.

Microsoft recommends that you either migrate any Azure Files or Table storage data to a separate storage account before you disallow access to the account via Shared Key, or that you do not apply this setting to storage accounts that support Azure Files or Table storage workloads.

Disallowing Shared Key access for a storage account does not affect SMB connections to Azure Files.

## Next steps

- [Authorize access to data in Azure Storage](#)
- [Authorize access to blobs and queues using Azure Active Directory](#)
- [Authorize with Shared Key](#)

# Create a user delegation SAS for a container or blob with PowerShell

8/22/2022 • 6 minutes to read • [Edit Online](#)

A shared access signature (SAS) enables you to grant limited access to containers and blobs in your storage account. When you create a SAS, you specify its constraints, including which Azure Storage resources a client is allowed to access, what permissions they have on those resources, and how long the SAS is valid.

Every SAS is signed with a key. You can sign a SAS in one of two ways:

- With a key created using Azure Active Directory (Azure AD) credentials. A SAS that is signed with Azure AD credentials is a *user delegation SAS*. A client that creates a user delegation SAS must be assigned an Azure RBAC role that includes the `Microsoft.Storage/storageAccounts/blobServices/generateUserDelegationKey` action. For more information, see [Create a user delegation SAS](#).
- With the storage account key. Both a *service SAS* and an *account SAS* are signed with the storage account key. The client that creates a service SAS must either have direct access to the account key or be assigned the `Microsoft.Storage/storageAccounts/listkeys/action` permission.

## NOTE

A user delegation SAS offers superior security to a SAS that is signed with the storage account key. Microsoft recommends using a user delegation SAS when possible. For more information, see [Grant limited access to data with shared access signatures \(SAS\)](#).

This article shows how to use Azure Active Directory (Azure AD) credentials to create a user delegation SAS for a container or blob with Azure PowerShell.

## About the user delegation SAS

A SAS token for access to a container or blob may be secured by using either Azure AD credentials or an account key. A SAS secured with Azure AD credentials is called a user delegation SAS, because the OAuth 2.0 token used to sign the SAS is requested on behalf of the user.

Microsoft recommends that you use Azure AD credentials when possible as a security best practice, rather than using the account key, which can be more easily compromised. When your application design requires shared access signatures, use Azure AD credentials to create a user delegation SAS for superior security. For more information about the user delegation SAS, see [Create a user delegation SAS](#).

### Caution

Any client that possesses a valid SAS can access data in your storage account as permitted by that SAS. It's important to protect a SAS from malicious or unintended use. Use discretion in distributing a SAS, and have a plan in place for revoking a compromised SAS.

For more information about shared access signatures, see [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#).

## Install the PowerShell module

To create a user delegation SAS with PowerShell, install version 1.10.0 or later of the Az.Storage module. Follow

these steps to install the latest version of the module:

1. Uninstall any previous installations of Azure PowerShell:
  - Remove any previous installations of Azure PowerShell from Windows using the **Apps & features** setting under **Settings**.
  - Remove all **Azure** modules from `%Program Files%\WindowsPowerShell\Modules`.
2. Make sure that you have the latest version of PowerShellGet installed. Open a Windows PowerShell window, and run the following command to install the latest version:

```
Install-Module PowerShellGet -Repository PSGallery -Force
```

3. Close and reopen the PowerShell window after installing PowerShellGet.

4. Install the latest version of Azure PowerShell:

```
Install-Module Az -Repository PSGallery -AllowClobber
```

5. Make sure that you have installed Azure PowerShell version 3.2.0 or later. Run the following command to install the latest version of the Azure Storage PowerShell module:

```
Install-Module -Name Az.Storage -Repository PSGallery -Force
```

6. Close and reopen the PowerShell window.

To check which version of the Az.Storage module is installed, run the following command:

```
Get-Module -ListAvailable -Name Az.Storage -Refresh
```

For more information about installing Azure PowerShell, see [Install Azure PowerShell with PowerShellGet](#).

## Sign in to Azure PowerShell with Azure AD

Call the [Connect-AzAccount](#) command to sign in with your Azure AD account:

```
Connect-AzAccount
```

For more information about signing in with PowerShell, see [Sign in with Azure PowerShell](#).

## Assign permissions with Azure RBAC

To create a user delegation SAS from Azure PowerShell, the Azure AD account used to sign into PowerShell must be assigned a role that includes the

**Microsoft.Storage/storageAccounts/blobServices/generateUserDelegationKey** action. This permission enables that Azure AD account to request the *user delegation key*. The user delegation key is used to sign the user delegation SAS. The role providing the

**Microsoft.Storage/storageAccounts/blobServices/generateUserDelegationKey** action must be assigned at the level of the storage account, the resource group, or the subscription. For more information about Azure RBAC permissions for creating a user delegation SAS, see the **Assign permissions with Azure RBAC** section in [Create a user delegation SAS](#).

If you do not have sufficient permissions to assign Azure roles to an Azure AD security principal, you may need

to ask the account owner or administrator to assign the necessary permissions.

The following example assigns the **Storage Blob Data Contributor** role, which includes the **Microsoft.Storage/storageAccounts/blobServices/generateUserDelegationKey** action. The role is scoped at the level of the storage account.

Remember to replace placeholder values in angle brackets with your own values:

```
New-AzRoleAssignment -SignInName <email> `
-RoleDefinitionName "Storage Blob Data Contributor" `
-Scope "/subscriptions/<subscription>/resourceGroups/<resource-
group>/providers/Microsoft.Storage/storageAccounts/<storage-account>"
```

For more information about the built-in roles that include the **Microsoft.Storage/storageAccounts/blobServices/generateUserDelegationKey** action, see [Azure built-in roles](#).

## Use Azure AD credentials to secure a SAS

When you create a user delegation SAS with Azure PowerShell, the user delegation key that is used to sign the SAS is created for you implicitly. The start time and expiry time that you specify for the SAS are also used as the start time and expiry time for the user delegation key.

Because the maximum interval over which the user delegation key is valid is 7 days from the start date, you should specify an expiry time for the SAS that is within 7 days of the start time. The SAS is invalid after the user delegation key expires, so a SAS with an expiry time of greater than 7 days will still only be valid for 7 days.

To create a user delegation SAS for a container or blob with Azure PowerShell, first create a new Azure Storage context object, specifying the `-UseConnectedAccount` parameter. The `-UseConnectedAccount` parameter specifies that the command creates the context object under the Azure AD account with which you signed in.

Remember to replace placeholder values in angle brackets with your own values:

```
$ctx = New-AzStorageContext -StorageAccountName <storage-account> -UseConnectedAccount
```

### Create a user delegation SAS for a container

To return a user delegation SAS token for a container, call the [New-AzStorageContainerSASToken](#) command, passing in the Azure Storage context object that you created previously.

The following example returns a user delegation SAS token for a container. Remember to replace the placeholder values in brackets with your own values:

```
New-AzStorageContainerSASToken -Context $ctx `
-Name <container> `
-Permission racwdl `
-ExpiryTime <date-time>
```

The user delegation SAS token returned will be similar to:

```
?sv=2018-11-09&sr=c&sig=<sig>&skoid=<skoid>&sktid=<sktid>&skt=2019-08-05T22%3A24%3A36Z&ske=2019-08-07T07%3A00%3A00Z&skbs=b&skv=2018-11-09&se=2019-08-07T07%3A00%3A00Z&sp=rwld
```

### Create a user delegation SAS for a blob

To return a user delegation SAS token for a blob, call the [New-AzStorageBlobSASToken](#) command, passing in the

Azure Storage context object that you created previously.

The following syntax returns a user delegation SAS for a blob. The example specifies the `-FullUri` parameter, which returns the blob URI with the SAS token appended. Remember to replace the placeholder values in brackets with your own values:

```
New-AzStorageBlobSASToken -Context $ctx `
 -Container <container> `
 -Blob <blob> `
 -Permission racwd `
 -ExpiryTime <date-time>
 -FullUri
```

The user delegation SAS URI returned will be similar to:

```
https://storagesamples.blob.core.windows.net/sample-container/blob1.txt?sv=2018-11-09&sr=b&sig=<sig>&skoid=<skoid>&sktid=<sktid>&skt=2019-08-06T21%3A16%3A54Z&ske=2019-08-07T07%3A00%3A00Z&sks=b&skv=2018-11-09&se=2019-08-07T07%3A00%3A00Z&sp=racwd
```

#### NOTE

A user delegation SAS does not support defining permissions with a stored access policy.

## Revoke a user delegation SAS

To revoke a user delegation SAS from Azure PowerShell, call the **Revoke-AzStorageAccountUserDelegationKeys** command. This command revokes all of the user delegation keys associated with the specified storage account. Any shared access signatures associated with those keys are invalidated.

Remember to replace placeholder values in angle brackets with your own values:

```
Revoke-AzStorageAccountUserDelegationKeys -ResourceGroupName <resource-group> `
 -StorageAccountName <storage-account>
```

#### IMPORTANT

Both the user delegation key and Azure role assignments are cached by Azure Storage, so there may be a delay between when you initiate the process of revocation and when an existing user delegation SAS becomes invalid.

## Next steps

- [Create a user delegation SAS \(REST API\)](#)
- [Get User Delegation Key operation](#)

# Create a user delegation SAS for a container or blob with the Azure CLI

8/22/2022 • 6 minutes to read • [Edit Online](#)

A shared access signature (SAS) enables you to grant limited access to containers and blobs in your storage account. When you create a SAS, you specify its constraints, including which Azure Storage resources a client is allowed to access, what permissions they have on those resources, and how long the SAS is valid.

Every SAS is signed with a key. You can sign a SAS in one of two ways:

- With a key created using Azure Active Directory (Azure AD) credentials. A SAS that is signed with Azure AD credentials is a *user delegation SAS*. A client that creates a user delegation SAS must be assigned an Azure RBAC role that includes the `Microsoft.Storage/storageAccounts/blobServices/generateUserDelegationKey` action. For more information, see [Create a user delegation SAS](#).
- With the storage account key. Both a *service SAS* and an *account SAS* are signed with the storage account key. The client that creates a service SAS must either have direct access to the account key or be assigned the `Microsoft.Storage/storageAccounts/listkeys/action` permission.

## NOTE

A user delegation SAS offers superior security to a SAS that is signed with the storage account key. Microsoft recommends using a user delegation SAS when possible. For more information, see [Grant limited access to data with shared access signatures \(SAS\)](#).

This article shows how to use Azure Active Directory (Azure AD) credentials to create a user delegation SAS for a container or blob with the Azure CLI.

## About the user delegation SAS

A SAS token for access to a container or blob may be secured by using either Azure AD credentials or an account key. A SAS secured with Azure AD credentials is called a user delegation SAS, because the OAuth 2.0 token used to sign the SAS is requested on behalf of the user.

Microsoft recommends that you use Azure AD credentials when possible as a security best practice, rather than using the account key, which can be more easily compromised. When your application design requires shared access signatures, use Azure AD credentials to create a user delegation SAS for superior security. For more information about the user delegation SAS, see [Create a user delegation SAS](#).

### Caution

Any client that possesses a valid SAS can access data in your storage account as permitted by that SAS. It's important to protect a SAS from malicious or unintended use. Use discretion in distributing a SAS, and have a plan in place for revoking a compromised SAS.

For more information about shared access signatures, see [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#).

## Install the latest version of the Azure CLI

To use the Azure CLI to secure a SAS with Azure AD credentials, first make sure that you have installed the latest

version of Azure CLI. For more information about installing the Azure CLI, see [Install the Azure CLI](#).

To create a user delegation SAS using the Azure CLI, make sure that you have installed version 2.0.78 or later. To check your installed version, use the `az --version` command.

## Sign in with Azure AD credentials

Sign in to the Azure CLI with your Azure AD credentials. For more information, see [Sign in with the Azure CLI](#).

## Assign permissions with Azure RBAC

To create a user delegation SAS from Azure PowerShell, the Azure AD account used to sign into Azure CLI must be assigned a role that includes the

`Microsoft.Storage/storageAccounts/blobServices/generateUserDelegationKey` action. This permission enables that Azure AD account to request the *user delegation key*. The user delegation key is used to sign the user delegation SAS. The role providing the

`Microsoft.Storage/storageAccounts/blobServices/generateUserDelegationKey` action must be assigned at the level of the storage account, the resource group, or the subscription.

If you do not have sufficient permissions to assign Azure roles to an Azure AD security principal, you may need to ask the account owner or administrator to assign the necessary permissions.

The following example assigns the **Storage Blob Data Contributor** role, which includes the

`Microsoft.Storage/storageAccounts/blobServices/generateUserDelegationKey` action. The role is scoped at the level of the storage account.

Remember to replace placeholder values in angle brackets with your own values:

```
az role assignment create \
--role "Storage Blob Data Contributor" \
--assignee <email> \
--scope "/subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>"
```

For more information about the built-in roles that include the

`Microsoft.Storage/storageAccounts/blobServices/generateUserDelegationKey` action, see [Azure built-in roles](#).

## Use Azure AD credentials to secure a SAS

When you create a user delegation SAS with the Azure CLI, the user delegation key that is used to sign the SAS is created for you implicitly. The start time and expiry time that you specify for the SAS are also used as the start time and expiry time for the user delegation key.

Because the maximum interval over which the user delegation key is valid is 7 days from the start date, you should specify an expiry time for the SAS that is within 7 days of the start time. The SAS is invalid after the user delegation key expires, so a SAS with an expiry time of greater than 7 days will still only be valid for 7 days.

When creating a user delegation SAS, the `--auth-mode login` and `--as-user parameters` are required. Specify `login` for the `--auth-mode` parameter so that requests made to Azure Storage are authorized with your Azure AD credentials. Specify the `--as-user` parameter to indicate that the SAS returned should be a user delegation SAS.

### Create a user delegation SAS for a container

To create a user delegation SAS for a container with the Azure CLI, call the `az storage container generate-sas` command.

Supported permissions for a user delegation SAS on a container include Add, Create, Delete, List, Read, and Write. Permissions can be specified singly or combined. For more information about these permissions, see [Create a user delegation SAS](#).

The following example returns a user delegation SAS token for a container. Remember to replace the placeholder values in brackets with your own values:

```
az storage container generate-sas \
--account-name <storage-account> \
--name <container> \
--permissions acdlrw \
--expiry <date-time> \
--auth-mode login \
--as-user
```

The user delegation SAS token returned will be similar to:

```
se=2019-07-27&sp=r&sv=2018-11-09&sr=c&skoid=<skoid>&sktid=<sktid>&skt=2019-07-26T18%3A01%3A22Z&ske=2019-07-27T00%3A00%3A00Z&skv=2018-11-09&sig=<signature>
```

## Create a user delegation SAS for a blob

To create a user delegation SAS for a blob with the Azure CLI, call the [az storage blob generate-sas](#) command.

Supported permissions for a user delegation SAS on a blob include Add, Create, Delete, Read, and Write. Permissions can be specified singly or combined. For more information about these permissions, see [Create a user delegation SAS](#).

The following syntax returns a user delegation SAS for a blob. The example specifies the `--full-uri` parameter, which returns the blob URI with the SAS token appended. Remember to replace the placeholder values in brackets with your own values:

```
az storage blob generate-sas \
--account-name <storage-account> \
--container-name <container> \
--name <blob> \
--permissions acdrw \
--expiry <date-time> \
--auth-mode login \
--as-user \
--full-uri
```

The user delegation SAS URI returned will be similar to:

```
https://storagesamples.blob.core.windows.net/sample-container/blob1.txt?se=2019-08-03&sp=rw&sv=2018-11-09&sr=b&skoid=<skoid>&sktid=<sktid>&skt=2019-08-02T2%3A32%3A01Z&ske=2019-08-03T00%3A00%3A00Z&skv=2018-11-09&sig=<signature>
```

### NOTE

A user delegation SAS does not support defining permissions with a stored access policy.

## Revoke a user delegation SAS

To revoke a user delegation SAS from the Azure CLI, call the [az storage account revoke-delegation-keys](#) command. This command revokes all of the user delegation keys associated with the specified storage account.

Any shared access signatures associated with those keys are invalidated.

Remember to replace placeholder values in angle brackets with your own values:

```
az storage account revoke-delegation-keys \
--name <storage-account> \
--resource-group <resource-group>
```

**IMPORTANT**

Both the user delegation key and Azure role assignments are cached by Azure Storage, so there may be a delay between when you initiate the process of revocation and when an existing user delegation SAS becomes invalid.

## Next steps

- [Create a user delegation SAS \(REST API\)](#)
- [Get User Delegation Key operation](#)

# Create a user delegation SAS for a container, directory, or blob with .NET

8/22/2022 • 10 minutes to read • [Edit Online](#)

A shared access signature (SAS) enables you to grant limited access to containers and blobs in your storage account. When you create a SAS, you specify its constraints, including which Azure Storage resources a client is allowed to access, what permissions they have on those resources, and how long the SAS is valid.

Every SAS is signed with a key. You can sign a SAS in one of two ways:

- With a key created using Azure Active Directory (Azure AD) credentials. A SAS that is signed with Azure AD credentials is a *user delegation SAS*. A client that creates a user delegation SAS must be assigned an Azure RBAC role that includes the `Microsoft.Storage/storageAccounts/blobServices/generateUserDelegationKey` action. For more information, see [Create a user delegation SAS](#).
- With the storage account key. Both a *service SAS* and an *account SAS* are signed with the storage account key. The client that creates a service SAS must either have direct access to the account key or be assigned the `Microsoft.Storage/storageAccounts/listkeys/action` permission.

## NOTE

A user delegation SAS offers superior security to a SAS that is signed with the storage account key. Microsoft recommends using a user delegation SAS when possible. For more information, see [Grant limited access to data with shared access signatures \(SAS\)](#).

This article shows how to use Azure Active Directory (Azure AD) credentials to create a user delegation SAS for a container, directory, or blob with the Azure Storage client library for .NET version 12.

## About the user delegation SAS

A SAS token for access to a container or blob may be secured by using either Azure AD credentials or an account key. A SAS secured with Azure AD credentials is called a user delegation SAS, because the OAuth 2.0 token used to sign the SAS is requested on behalf of the user.

Microsoft recommends that you use Azure AD credentials when possible as a security best practice, rather than using the account key, which can be more easily compromised. When your application design requires shared access signatures, use Azure AD credentials to create a user delegation SAS for superior security. For more information about the user delegation SAS, see [Create a user delegation SAS](#).

### Caution

Any client that possesses a valid SAS can access data in your storage account as permitted by that SAS. It's important to protect a SAS from malicious or unintended use. Use discretion in distributing a SAS, and have a plan in place for revoking a compromised SAS.

For more information about shared access signatures, see [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#).

## Assign Azure roles for access to data

When an Azure AD security principal attempts to access blob data, that security principal must have permissions

to the resource. Whether the security principal is a managed identity in Azure or an Azure AD user account running code in the development environment, the security principal must be assigned an Azure role that grants access to blob data in Azure Storage. For information about assigning permissions via Azure RBAC, see [Assign an Azure role for access to blob data](#).

To learn more about how to get a token with the Azure Identity client library from Azure Storage, see [Use the Azure Identity library to get an access token for authorization](#).

## Get an authenticated token credential

To get a token credential that your code can use to authorize requests to Azure Storage, create an instance of the [DefaultAzureCredential](#) class. For more information about using the DefaultAzureCredential class to authorize a managed identity to access Azure Storage, see [Azure Identity client library for .NET](#).

The following code snippet shows how to get the authenticated token credential and use it to create a service client for Blob storage:

```
// Construct the blob endpoint from the account name.
string blobEndpoint = string.Format("https://'{0}'.blob.core.windows.net", accountName);

// Create a new Blob service client with Azure AD credentials.
BlobServiceClient blobClient = new BlobServiceClient(new Uri(blobEndpoint),
 new DefaultAzureCredential());
```

## Get the user delegation key

Every SAS is signed with a key. To create a user delegation SAS, you must first request a user delegation key, which is then used to sign the SAS. The user delegation key is analogous to the account key used to sign a service SAS or an account SAS, except that it relies on your Azure AD credentials. When a client requests a user delegation key using an OAuth 2.0 token, Azure Storage returns the user delegation key on behalf of the user.

Once you have the user delegation key, you can use that key to create any number of user delegation shared access signatures, over the lifetime of the key. The user delegation key is independent of the OAuth 2.0 token used to acquire it, so the token does not need to be renewed so long as the key is still valid. You can specify that the key is valid for a period of up to 7 days.

Use one of the following methods to request the user delegation key:

- [GetUserDelegationKey](#)
- [GetUserDelegationKeyAsync](#)

The following code snippet gets the user delegation key and writes out its properties:

```
// Get a user delegation key for the Blob service that's valid for seven days.
// You can use the key to generate any number of shared access signatures over the lifetime of the key.
UserDelegationKey key = await blobClient.GetUserDelegationKeyAsync(DateTimeOffset.UtcNow,
 DateTimeOffset.UtcNow.AddDays(7));

// Read the key's properties.
Console.WriteLine("User delegation key properties:");
Console.WriteLine("Key signed start: {0}", key.SignedStartsOn);
Console.WriteLine("Key signed expiry: {0}", key.SignedExpiresOn);
Console.WriteLine("Key signed object ID: {0}", key.SignedObjectId);
Console.WriteLine("Key signed tenant ID: {0}", key.SignedTenantId);
Console.WriteLine("Key signed service: {0}", key.SignedService);
Console.WriteLine("Key signed version: {0}", key.SignedVersion);
```

## Get a user delegation SAS for a blob

The following code example shows the complete code for authenticating the security principal and creating the user delegation SAS for a blob:

```
async static Task<Uri> GetUserDelegationSasBlob(BlobClient blobClient)
{
 BlobServiceClient blobServiceClient =
 blobClient.GetParentBlobContainerClient().GetParentBlobServiceClient();

 // Get a user delegation key for the Blob service that's valid for 7 days.
 // You can use the key to generate any number of shared access signatures
 // over the lifetime of the key.
 Azure.Storage.Blobs.Models.UserDelegationKey userDelegationKey =
 await blobServiceClient.GetUserDelegationKeyAsync(DateTimeOffset.UtcNow,
 DateTimeOffset.UtcNow.AddDays(7));

 // Create a SAS token that's also valid for 7 days.
 BlobSasBuilder sasBuilder = new BlobSasBuilder()
 {
 BlobContainerName = blobClient.BlobContainerName,
 BlobName = blobClient.Name,
 Resource = "b",
 StartsOn = DateTimeOffset.UtcNow,
 ExpiresOn = DateTimeOffset.UtcNow.AddDays(7)
 };

 // Specify read and write permissions for the SAS.
 sasBuilder.SetPermissions(BlobSasPermissions.Read |
 BlobSasPermissions.Write);

 // Add the SAS token to the blob URI.
 BlobUriBuilder blobUriBuilder = new BlobUriBuilder(blobClient.Uri)
 {
 // Specify the user delegation key.
 Sas = sasBuilder.ToSasQueryParameters(userDelegationKey,
 blobServiceClient.AccountName)
 };

 Console.WriteLine("Blob user delegation SAS URI: {0}", blobUriBuilder);
 Console.WriteLine();
 return blobUriBuilder.ToUri();
}
```

The following example tests the user delegation SAS created in the previous example from a simulated client application. If the SAS is valid, the client application is able to read the contents of the blob. If the SAS is invalid, for example if it has expired, Azure Storage returns error code 403 (Forbidden).

```

static async Task ReadBlobWithSasAsync(Uri sasUri)
{
 // Try performing a read operation using the blob SAS provided.

 // Create a blob client object for blob operations.
 BlobClient blobClient = new BlobClient(sasUri, null);

 // Download and read the contents of the blob.
 try
 {
 Console.WriteLine("Blob contents:");

 // Download blob contents to a stream and read the stream.
 BlobDownloadInfo blobDownloadInfo = await blobClient.DownloadAsync();
 using (StreamReader reader = new StreamReader(blobDownloadInfo.Content, true))
 {
 string line;
 while ((line = reader.ReadLine()) != null)
 {
 Console.WriteLine(line);
 }
 }

 Console.WriteLine();
 Console.WriteLine("Read operation succeeded for SAS {0}", sasUri);
 Console.WriteLine();
 }
 catch (RequestFailedException e)
 {
 // Check for a 403 (Forbidden) error. If the SAS is invalid,
 // Azure Storage returns this error.
 if (e.Status == 403)
 {
 Console.WriteLine("Read operation failed for SAS {0}", sasUri);
 Console.WriteLine("Additional error information: " + e.Message);
 Console.WriteLine();
 }
 else
 {
 Console.WriteLine(e.Message);
 Console.ReadLine();
 throw;
 }
 }
}
}

```

## Get a user delegation SAS for a container

The following code example shows how to generate a user delegation SAS for a container:

```

async static Task<Uri> GetUserDelegationSasContainer(BlobContainerClient blobContainerClient)
{
 BlobServiceClient blobServiceClient = blobContainerClient.GetParentBlobServiceClient();

 // Get a user delegation key for the Blob service that's valid for seven days.
 // You can use the key to generate any number of shared access signatures
 // over the lifetime of the key.
 Azure.Storage.Blobs.Models.UserDelegationKey userDelegationKey =
 await blobServiceClient.GetUserDelegationKeyAsync(DateTimeOffset.UtcNow,
 DateTimeOffset.UtcNow.AddDays(7));

 // Create a SAS token that's also valid for seven days.
 BlobSasBuilder sasBuilder = new BlobSasBuilder()
 {
 BlobContainerName = blobContainerClient.Name,
 Resource = "c",
 StartsOn = DateTimeOffset.UtcNow,
 ExpiresOn = DateTimeOffset.UtcNow.AddDays(7)
 };

 // Specify racwl permissions for the SAS.
 sasBuilder.SetPermissions(
 BlobContainerSasPermissions.Read |
 BlobContainerSasPermissions.Add |
 BlobContainerSasPermissions.Create |
 BlobContainerSasPermissions.Write |
 BlobContainerSasPermissions.List
);

 // Add the SAS token to the container URI.
 BlobUriBuilder blobUriBuilder = new BlobUriBuilder(blobContainerClient.Uri)
 {
 // Specify the user delegation key.
 Sas = sasBuilder.ToSasQueryParameters(userDelegationKey,
 blobServiceClient.AccountName)
 };

 Console.WriteLine("Container user delegation SAS URI: {0}", blobUriBuilder);
 Console.WriteLine();
 return blobUriBuilder.ToUri();
}

```

The following example tests the user delegation SAS created in the previous example from a simulated client application. If the SAS is valid, the client application is able to read the contents of the blob. If the SAS is invalid, for example if it has expired, Azure Storage returns error code 403 (Forbidden).

```

private static async Task ListBlobsWithSasAsync(Uri sasUri)
{
 // Try performing a listing operation using the container SAS provided.

 // Create a container client object for blob operations.
 BlobContainerClient blobContainerClient = new BlobContainerClient(sasUri, null);

 // List blobs in the container.
 try
 {
 // Call the listing operation and return pages of the specified size.
 var resultSegment = blobContainerClient.GetBlobsAsync().AsPages();

 // Enumerate the blobs returned for each page.
 await foreach (Azure.Page<BlobItem> blobPage in resultSegment)
 {
 foreach (BlobItem blobItem in blobPage.Values)
 {
 Console.WriteLine("Blob name: {0}", blobItem.Name);
 }
 Console.WriteLine();
 }

 Console.WriteLine();
 Console.WriteLine("Blob listing operation succeeded for SAS {0}", sasUri);
 }
 catch (RequestFailedException e)
 {
 // Check for a 403 (Forbidden) error. If the SAS is invalid,
 // Azure Storage returns this error.
 if (e.Status == 403)
 {
 Console.WriteLine("Blob listing operation failed for SAS {0}", sasUri);
 Console.WriteLine("Additional error information: " + e.Message);
 Console.WriteLine();
 }
 else
 {
 Console.WriteLine(e.Message);
 Console.ReadLine();
 throw;
 }
 }
}

```

## Get a user delegation SAS for a directory

The following code example shows how to generate a user delegation SAS for a directory when a hierarchical namespace is enabled for the storage account:

```

async static Task<Uri> GetUserDelegationSasDirectory(DataLakeDirectoryClient directoryClient)
{
 try
 {
 // Get service endpoint from the directory URI.
 DataLakeUriBuilder dataLakeServiceUri = new DataLakeUriBuilder(directoryClient.Uri)
 {
 FileSystemName = null,
 DirectoryOrFilePath = null
 };

 // Get service client.
 DataLakeServiceClient dataLakeServiceClient =
 new DataLakeServiceClient(dataLakeServiceUri.ToUri(),
 new DefaultAzureCredential());

 // Get a user delegation key that's valid for seven days.
 // You can use the key to generate any number of shared access signatures
 // over the lifetime of the key.
 Azure.Storage.Files.DataLake.Models.UserDelegationKey userDelegationKey =
 await dataLakeServiceClient.GetUserDelegationKeyAsync(DateTimeOffset.UtcNow,
 DateTimeOffset.UtcNow.AddDays(7));

 // Create a SAS token that's valid for seven days.
 DataLakeSasBuilder sasBuilder = new DataLakeSasBuilder()
 {
 // Specify the file system name and path, and indicate that
 // the client object points to a directory.
 FileSystemName = directoryClient.FileSystemName,
 Resource = "d",
 IsDirectory = true,
 Path = directoryClient.Path,
 ExpiresOn = DateTimeOffset.UtcNow.AddDays(7)
 };

 // Specify racwl permissions for the SAS.
 sasBuilder.SetPermissions(
 DataLakeSasPermissions.Read |
 DataLakeSasPermissions.Add |
 DataLakeSasPermissions.Create |
 DataLakeSasPermissions.Write |
 DataLakeSasPermissions.List
);

 // Construct the full URI, including the SAS token.
 DataLakeUriBuilder fullUri = new DataLakeUriBuilder(directoryClient.Uri)
 {
 Sas = sasBuilder.ToSasQueryParameters(userDelegationKey,
 dataLakeServiceClient.AccountName)
 };

 Console.WriteLine("Directory user delegation SAS URI: {0}", fullUri);
 Console.WriteLine();
 return fullUri.ToUri();
 }
 catch (Exception e)
 {
 Console.WriteLine(e.Message);
 throw;
 }
}

```

The following example tests the user delegation SAS created in the previous example from a simulated client application. If the SAS is valid, the client application is able to list file paths for this directory. If the SAS is invalid, for example if it has expired, Azure Storage returns error code 403 (Forbidden).

```

private static async Task ListFilePathsWithDirectorySasAsync(Uri sasUri)
{
 // Try performing an operation using the directory SAS provided.

 // Create a directory client object for listing operations.
 DataLakeDirectoryClient dataLakeDirectoryClient = new DataLakeDirectoryClient(sasUri);

 // List file paths in the directory.
 try
 {
 // Call the listing operation and return pages of the specified size.
 var resultSegment = dataLakeDirectoryClient.GetPathsAsync(false, false).AsPages();

 // Enumerate the file paths returned with each page.
 await foreach (Page<PathItem> pathPage in resultSegment)
 {
 foreach (PathItem pathItem in pathPage.Values)
 {
 Console.WriteLine("File name: {0}", pathItem.Name);
 }
 Console.WriteLine();
 }

 Console.WriteLine();
 Console.WriteLine("Directory listing operation succeeded for SAS {0}", sasUri);
 }
 catch (RequestFailedException e)
 {
 // Check for a 403 (Forbidden) error. If the SAS is invalid,
 // Azure Storage returns this error.
 if (e.Status == 403)
 {
 Console.WriteLine("Directory listing operation failed for SAS {0}", sasUri);
 Console.WriteLine("Additional error information: " + e.Message);
 Console.WriteLine();
 }
 else
 {
 Console.WriteLine(e.Message);
 Console.ReadLine();
 throw;
 }
 }
}

```

## Resources for development with .NET

The links below provide useful resources for developers using the Azure Storage client library for .NET.

### Azure Storage common APIs

- [API reference documentation](#)
- [Library source code](#)
- [Package \(NuGet\)](#)

### Blob storage APIs

- [API reference documentation](#)
- [Library source code](#)
- [Package \(NuGet\) for version 11.x](#)
- [Package \(NuGet\) for version 12.x](#)
- [Samples](#)

## .NET tools

- [.NET](#)
- [Visual Studio](#)

## See also

- [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#)
- [Get User Delegation Key operation](#)
- [Create a user delegation SAS \(REST API\)](#)

# Create a service SAS for a container or blob

8/22/2022 • 8 minutes to read • [Edit Online](#)

A shared access signature (SAS) enables you to grant limited access to containers and blobs in your storage account. When you create a SAS, you specify its constraints, including which Azure Storage resources a client is allowed to access, what permissions they have on those resources, and how long the SAS is valid.

Every SAS is signed with a key. You can sign a SAS in one of two ways:

- With a key created using Azure Active Directory (Azure AD) credentials. A SAS that is signed with Azure AD credentials is a *user delegation* SAS. A client that creates a user delegation SAS must be assigned an Azure RBAC role that includes the **Microsoft.Storage/storageAccounts/blobServices/generateUserDelegationKey** action. For more information, see [Create a user delegation SAS](#).
- With the storage account key. Both a *service SAS* and an *account SAS* are signed with the storage account key. The client that creates a service SAS must either have direct access to the account key or be assigned the **Microsoft.Storage/storageAccounts/listkeys/action** permission.

## NOTE

A user delegation SAS offers superior security to a SAS that is signed with the storage account key. Microsoft recommends using a user delegation SAS when possible. For more information, see [Grant limited access to data with shared access signatures \(SAS\)](#).

This article shows how to use the storage account key to create a service SAS for a container or blob with the Azure Storage client library for Blob Storage.

## Create a service SAS for a blob container

The following code example creates a SAS for a container. If the name of an existing stored access policy is provided, that policy is associated with the SAS. If no stored access policy is provided, then the code creates an ad hoc SAS on the container.

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)
- [JavaScript v12 SDK](#)

A service SAS is signed with the account access key. Use the **StorageSharedKeyCredential** class to create the credential that is used to sign the SAS. Next, create a new **BlobSasBuilder** object and call the **ToSasQueryParameters** to get the SAS token string.

```

private static Uri GetServiceSasUriForContainer(BlobContainerClient containerClient,
 string storedPolicyName = null)
{
 // Check whether this BlobContainerClient object has been authorized with Shared Key.
 if (containerClient.CanGenerateSasUri)
 {
 // Create a SAS token that's valid for one hour.
 BlobSasBuilder sasBuilder = new BlobSasBuilder()
 {
 BlobContainerName = containerClient.Name,
 Resource = "c"
 };

 if (storedPolicyName == null)
 {
 sasBuilder.ExpiresOn = DateTimeOffset.UtcNow.AddHours(1);
 sasBuilder.SetPermissions(BlobContainerSasPermissions.Read);
 }
 else
 {
 sasBuilder.Identifier = storedPolicyName;
 }

 Uri sasUri = containerClient.GenerateSasUri(sasBuilder);
 Console.WriteLine("SAS URI for blob container is: {0}", sasUri);
 Console.WriteLine();

 return sasUri;
 }
 else
 {
 Console.WriteLine(@"BlobContainerClient must be authorized with Shared Key
credentials to create a service SAS.");
 return null;
 }
}

```

## Create a service SAS for a blob

The following code example creates a SAS on a blob. If the name of an existing stored access policy is provided, that policy is associated with the SAS. If no stored access policy is provided, then the code creates an ad hoc SAS on the blob.

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)
- [JavaScript v12 SDK](#)

A service SAS is signed with the account access key. Use the [StorageSharedKeyCredential](#) class to create the credential that is used to sign the SAS. Next, create a new [BlobSasBuilder](#) object and call the [ToSasQueryParameters](#) to get the SAS token string.

```

private static Uri GetServiceSasUriForBlob(BlobClient blobClient,
 string storedPolicyName = null)
{
 // Check whether this BlobClient object has been authorized with Shared Key.
 if (blobClient.CanGenerateSasUri)
 {
 // Create a SAS token that's valid for one hour.
 BlobSasBuilder sasBuilder = new BlobSasBuilder()
 {
 BlobContainerName = blobClient.GetParentBlobContainerClient().Name,
 BlobName = blobClient.Name,
 Resource = "b"
 };

 if (storedPolicyName == null)
 {
 sasBuilder.ExpiresOn = DateTimeOffset.UtcNow.AddHours(1);
 sasBuilder.SetPermissions(BlobSasPermissions.Read |
 BlobSasPermissions.Write);
 }
 else
 {
 sasBuilder.Identifier = storedPolicyName;
 }

 Uri sasUri = blobClient.GenerateSasUri(sasBuilder);
 Console.WriteLine("SAS URI for blob is: {0}", sasUri);
 Console.WriteLine();

 return sasUri;
 }
 else
 {
 Console.WriteLine(@"BlobClient must be authorized with Shared Key
 credentials to create a service SAS.");
 return null;
 }
}

```

## Create a service SAS for a directory

In a storage account with a hierarchical namespace enabled, you can create a service SAS for a directory. To create the service SAS, make sure you have installed version 12.5.0 or later of the [Azure.Storage.Files.DataLake](#) package.

The following example shows how to create a service SAS for a directory with the v12 client library for .NET:

```

private static Uri GetServiceSasUriForDirectory(DataLakeDirectoryClient directoryClient,
 string storedPolicyName = null)
{
 if (directoryClient.CanGenerateSasUri)
 {
 // Create a SAS token that's valid for one hour.
 DataLakeSasBuilder sasBuilder = new DataLakeSasBuilder()
 {
 // Specify the file system name, the path, and indicate that
 // the client object points to a directory.
 FileSystemName = directoryClient.FileSystemName,
 Resource = "d",
 IsDirectory = true,
 Path = directoryClient.Path,
 };

 // If no stored access policy is specified, create the policy
 // by specifying expiry and permissions.
 if (storedPolicyName == null)
 {
 sasBuilder.ExpiresOn = DateTimeOffset.UtcNow.AddHours(1);
 sasBuilder.SetPermissions(DataLakePermissions.Read |
 DataLakePermissions.Write |
 DataLakePermissions.List);
 }
 else
 {
 sasBuilder.Identifier = storedPolicyName;
 }

 // Get the SAS URI for the specified directory.
 Uri sasUri = directoryClient.GenerateSasUri(sasBuilder);
 Console.WriteLine("SAS URI for ADLS directory is: {0}", sasUri);
 Console.WriteLine();

 return sasUri;
 }
 else
 {
 Console.WriteLine(@"DataLakeDirectoryClient must be authorized with Shared Key
 credentials to create a service SAS.");
 return null;
 }
}

```

## Resources for development with .NET

The links below provide useful resources for developers using the Azure Storage client library for .NET.

### Azure Storage common APIs

- [API reference documentation](#)
- [Library source code](#)
- [Package \(NuGet\)](#)

### Blob storage APIs

- [API reference documentation](#)
- [Library source code](#)
- [Package \(NuGet\) for version 11.x](#)
- [Package \(NuGet\) for version 12.x](#)
- [Samples](#)

## .NET tools

- [.NET](#)
- [Visual Studio](#)

# Resources for development with JavaScript

The links below provide useful resources for developers using the Azure Storage client library for JavaScript

## Blob storage APIs

- [Azure Storage Blob client library for JavaScript](#)
- [Library source code](#)
- [Package \(npm\)](#)
- [API reference documentation](#)

## JavaScript tools

- [Node.js](#)
- [Visual Studio Code](#)

## Next steps

- [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#)
- [Create a service SAS](#)

# Create an account SAS with .NET

8/22/2022 • 4 minutes to read • [Edit Online](#)

A shared access signature (SAS) enables you to grant limited access to containers and blobs in your storage account. When you create a SAS, you specify its constraints, including which Azure Storage resources a client is allowed to access, what permissions they have on those resources, and how long the SAS is valid.

Every SAS is signed with a key. You can sign a SAS in one of two ways:

- With a key created using Azure Active Directory (Azure AD) credentials. A SAS that is signed with Azure AD credentials is a *user delegation* SAS. A client that creates a user delegation SAS must be assigned an Azure RBAC role that includes the `Microsoft.Storage/storageAccounts/blobServices/generateUserDelegationKey` action. For more information, see [Create a user delegation SAS](#).
- With the storage account key. Both a *service SAS* and an *account SAS* are signed with the storage account key. The client that creates a service SAS must either have direct access to the account key or be assigned the `Microsoft.Storage/storageAccounts/listkeys/action` permission.

## NOTE

A user delegation SAS offers superior security to a SAS that is signed with the storage account key. Microsoft recommends using a user delegation SAS when possible. For more information, see [Grant limited access to data with shared access signatures \(SAS\)](#).

This article shows how to use the storage account key to create an account SAS with the [Azure Storage client library for .NET](#).

## Create an account SAS

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

An account SAS is signed with the account access key. Use the `StorageSharedKeyCredential` class to create the credential that is used to sign the SAS. Next, create a new `AccountSasBuilder` object and call the `ToSasQueryParameters` to get the SAS token string.

```
private static string GetAccountSASToken(StorageSharedKeyCredential key)
{
 // Create a SAS token that's valid for one hour.
 AccountSasBuilder sasBuilder = new AccountSasBuilder()
 {
 Services = AccountSasServices.Blobs | AccountSasServices.Files,
 ResourceTypes = AccountSasResourceTypes.Service,
 ExpiresOn = DateTimeOffset.UtcNow.AddHours(1),
 Protocol = SasProtocol.Https
 };

 sasBuilder.SetPermissions(AccountSasPermissions.Read |
 AccountSasPermissions.Write);

 // Use the key to get the SAS token.
 string sasToken = sasBuilder.ToSasQueryParameters(key).ToString();

 Console.WriteLine("SAS token for the storage account is: {0}", sasToken);
 Console.WriteLine();

 return sasToken;
}
```

## Use an account SAS from a client

To use the account SAS to access service-level APIs for the Blob service, construct a Blob service client object using the SAS and the Blob storage endpoint for your storage account.

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

```

private static void UseAccountSAS(Uri blobServiceUri, string sasToken)
{
 var blobServiceClient = new BlobServiceClient
 (new Uri($"{blobServiceUri}?{sasToken}"), null);

 BlobRetentionPolicy retentionPolicy = new BlobRetentionPolicy();
 retentionPolicy.Enabled = true;
 retentionPolicy.Days = 7;

 blobServiceClient.SetProperties(new BlobServiceProperties()
 {
 HourMetrics = new BlobMetrics()
 {
 RetentionPolicy = retentionPolicy,
 Version = "1.0"
 },
 MinuteMetrics = new BlobMetrics()
 {
 RetentionPolicy = retentionPolicy,
 Version = "1.0"
 },
 Logging = new BlobAnalyticsLogging()
 {
 Write = true,
 Read = true,
 Delete = true,
 RetentionPolicy = retentionPolicy,
 Version = "1.0"
 }
 });
}

// The permissions granted by the account SAS also permit you to retrieve service properties.

BlobServiceProperties serviceProperties = blobServiceClient.GetProperties().Value;
Console.WriteLine(serviceProperties.HourMetrics.RetentionPolicy);
Console.WriteLine(serviceProperties.HourMetrics.Version);
}

```

## Next steps

- [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#)
- [Create an account SAS](#)

# Create a stored access policy with .NET

8/22/2022 • 3 minutes to read • [Edit Online](#)

A stored access policy provides an additional level of control over service-level shared access signatures (SAS) on the server side. Defining a stored access policy serves to group shared access signatures and to provide additional restrictions for shared access signatures that are bound by the policy. You can use a stored access policy to change the start time, expiry time, or permissions for a SAS, or to revoke it after it has been issued.

The following Azure Storage resources support stored access policies:

- Blob containers
- File shares
- Queues
- Tables

## NOTE

A stored access policy on a container can be associated with a shared access signature granting permissions to the container itself or to the blobs it contains. Similarly, a stored access policy on a file share can be associated with a shared access signature granting permissions to the share itself or to the files it contains.

Stored access policies are supported for a service SAS only. Stored access policies are not supported for account SAS or user delegation SAS.

For more information about stored access policies, see [Define a stored access policy](#).

## Create a stored access policy

The underlying REST operation to create a stored access policy is [Set Container ACL](#). You must authorize the operation to create a stored access policy via Shared Key by using the account access keys in a connection string. Authorizing the **Set Container ACL** operation with Azure AD credentials is not supported. For more information, see [Permissions for calling data operations](#).

The following code examples create a stored access policy on a container. You can use the access policy to specify constraints for a service SAS on the container or its blobs.

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

To create a stored access policy on a container with version 12 of the .NET client library for Azure Storage, call one of the following methods:

- [BlobContainerClient.SetAccessPolicy](#)
- [BlobContainerClient.SetAccessPolicyAsync](#)

The following example creates a stored access policy that is in effect for one day and that grants read/write permissions:

```

async static Task CreateStoredAccessPolicyAsync(string containerName)
{
 string connectionString = "";

 // Use the connection string to authorize the operation to create the access policy.
 // Azure AD does not support the Set Container ACL operation that creates the policy.
 BlobContainerClient containerClient = new BlobContainerClient(connectionString, containerName);

 try
 {
 await containerClient.CreateIfNotExistsAsync();

 // Create one or more stored access policies.
 List<BlobSignedIdentifier> signedIdentifiers = new List<BlobSignedIdentifier>
 {
 new BlobSignedIdentifier
 {
 Id = "mysignedidentifier",
 AccessPolicy = new BlobAccessPolicy
 {
 StartsOn = DateTimeOffset.UtcNow.AddHours(-1),
 ExpiresOn = DateTimeOffset.UtcNow.AddDays(1),
 Permissions = "rw"
 }
 }
 };
 // Set the container's access policy.
 await containerClient.SetAccessPolicyAsync(permissions: signedIdentifiers);
 }
 catch (RequestFailedException e)
 {
 Console.WriteLine(e.ErrorCode);
 Console.WriteLine(e.Message);
 }
 finally
 {
 await containerClient.DeleteAsync();
 }
}

```

## See also

- [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#)
- [Define a stored access policy](#)
- [Configure Azure Storage connection strings](#)

# Create an expiration policy for shared access signatures

8/22/2022 • 6 minutes to read • [Edit Online](#)

You can use a shared access signature (SAS) to delegate access to resources in your Azure Storage account. A SAS token includes the targeted resource, the permissions granted, and the interval over which access is permitted. Best practices recommend that you limit the interval for a SAS in case it is compromised. By setting a SAS expiration policy for your storage accounts, you can provide a recommended upper expiration limit when a user creates a service SAS or an account SAS.

For more information about shared access signatures, see [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#).

## About SAS expiration policies

You can configure a SAS expiration policy on the storage account. The SAS expiration policy specifies the recommended upper limit for the signed expiry field on a service SAS or an account SAS. The recommended upper limit is specified as a date/time value that is a combined number of days, hours, minutes, and seconds.

The validity interval for the SAS is calculated by subtracting the date/time value of the signed start field from the date/time value of the signed expiry field. If the resulting value is less than or equal to the recommended upper limit, then the SAS is in compliance with the SAS expiration policy.

After you configure the SAS expiration policy, then a user who creates a SAS with an interval that exceeds the recommended upper limit will see a warning.

A SAS expiration policy does not prevent a user from creating a SAS with an expiration that exceeds the limit recommended by the policy. When a user creates a SAS that violates the policy, they'll see a warning, together with the recommended maximum interval. If you have configured a diagnostic setting for logging with Azure Monitor, then Azure Storage writes a message to the `SasExpiryStatus` property in the logs whenever a user creates or uses a SAS that expires after the recommended interval. The message indicates that the validity interval of the SAS exceeds the recommended interval.

When a SAS expiration policy is in effect for the storage account, the signed start field is required for every SAS. If the signed start field is not included on the SAS, and you have configured a diagnostic setting for logging with Azure Monitor, then Azure Storage writes a message to the `SasExpiryStatus` property in the logs whenever a user creates or uses a SAS without a value for the signed start field.

## Create a SAS expiration policy

When you create a SAS expiration policy on a storage account, the policy applies to each type of SAS that is signed with the account key. The types of shared access signatures that are signed with the account key are the service SAS and the account SAS.

### NOTE

Before you can create a SAS expiration policy, you may need to rotate each of your account access keys at least once.

- [Azure portal](#)
- [PowerShell](#)

- Azure CLI

To create a SAS expiration policy in the Azure portal, follow these steps:

1. Navigate to your storage account in the Azure portal.
2. Under **Settings**, select **Configuration**.
3. Locate the setting for **Allow recommended upper limit for shared access signature (SAS) expiry interval**, and set it to **Enabled**. If the setting appears disabled, then you need to rotate both account access keys before you can set a recommended upper limit for SAS expiry interval.
4. Specify the recommended interval for any new shared access signatures that are created on resources in this storage account.

The screenshot shows the Azure Storage Configuration page for the 'storagesamplessas' storage account. The left sidebar lists various settings like Blob inventory, Static website, Lifecycle management, and Configuration (which is selected). The main pane shows account details such as kind (StorageV2), performance (Standard), and security settings (Secure transfer required, Allow Blob public access, Allow storage account key access). A red box highlights the 'Allow recommended upper limit for shared access signature (SAS) expiry interval' section, which is currently set to 'Enabled'. Below this, there are input fields for 'Days', 'Hours', 'Minutes', and 'Seconds' to specify the recommended upper limit for SAS expiry interval. Other settings shown include 'Default to Azure Active Directory authorization in the Azure portal' (disabled) and 'Minimum TLS version' (Version 1.2).

5. Select the **Save** button to save your changes.

## Query logs for policy violations

To log the creation of a SAS that is valid over a longer interval than the SAS expiration policy recommends, first create a diagnostic setting that sends logs to an Azure Log Analytics workspace. For more information, see [Send logs to Azure Log Analytics](#).

Next, use an Azure Monitor log query to monitor whether policy has been violated. Create a new query in your Log Analytics workspace, add the following query text, and press **Run**.

```
StorageBlobLogs
| where SasExpiryStatus startswith "Policy violated"
| summarize count() by AccountName, SasExpiryStatus
```

## Use a built-in policy to monitor compliance

You can monitor your storage accounts with Azure Policy to ensure that storage accounts in your subscription have configured SAS expiration policies. Azure Storage provides a built-in policy for ensuring that accounts have this setting configured. For more information about the built-in policy, see **Storage accounts should have shared access signature (SAS) policies configured** in [List of built-in policy definitions](#).

### Assign the built-in policy for a resource scope

Follow these steps to assign the built-in policy to the appropriate scope in the Azure portal:

1. In the Azure portal, search for *Policy* to display the Azure Policy dashboard.
2. In the **Authoring** section, select **Assignments**.
3. Choose **Assign policy**.
4. On the **Basics** tab of the **Assign policy** page, in the **Scope** section, specify the scope for the policy assignment. Select the **More** button to choose the subscription and optional resource group.
5. For the **Policy definition** field, select the **More** button, and enter *storage account keys* in the **Search** field. Select the policy definition named **Storage account keys should not be expired**.

### Available Definitions

X

Type	Search
All types	shared access signature

#### Policy Definitions (1)

##### Storage accounts should have shared access signature (SAS) policies configured

Built-in

Ensure storage accounts have shared access signature (SAS) expiration policy enabled. Users use a SAS to delegate access to resources in Azure Storage account. And SAS expiration policy recommend upper expiration limit when a user creates a SAS token.

Select

Cancel

6. Select **Review + create** to assign the policy definition to the specified scope.

# Assign policy

Basics Parameters Remediation Non-compliance messages Review + create

## Scope

Scope [Learn more about setting the scope \\*](#)

Azure Storage content development and testing/storage-resource-group-create



## Exclusions

Optionally select resources to exclude from the policy assignment.



## Basics

Policy definition \*

Storage accounts should have shared access signature (SAS) policies configured



Assignment name \* ⓘ

Storage accounts should have shared access signature (SAS) policies configured



## Description

Policy enforcement ⓘ

Enabled  Disabled

Assigned by

[Review + create](#)

[Cancel](#)

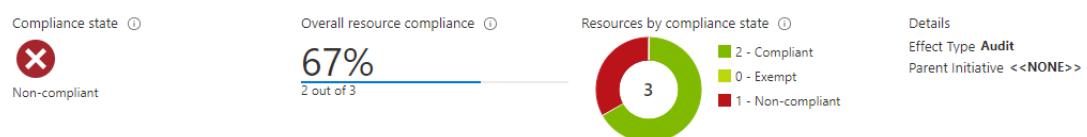
[Previous](#)

[Next](#)

## Monitor compliance with the key expiration policy

To monitor your storage accounts for compliance with the key expiration policy, follow these steps:

1. On the Azure Policy dashboard, locate the built-in policy definition for the scope that you specified in the policy assignment. You can search for *Storage accounts should have shared access signature (SAS) policies configured* in the **Search** box to filter for the built-in policy.
2. Select the policy name with the desired scope.
3. On the **Policy assignment** page for the built-in policy, select **View compliance**. Any storage accounts in the specified subscription and resource group that do not meet the policy requirements appear in the compliance report.



To bring a storage account into compliance, configure a SAS expiration policy for that account, as described in

[Create a SAS expiration policy.](#)

## See also

- [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#)
- [Create a service SAS](#)
- [Create an account SAS](#)

# Configure anonymous public read access for containers and blobs

8/22/2022 • 12 minutes to read • [Edit Online](#)

Azure Storage supports optional anonymous public read access for containers and blobs. By default, anonymous access to your data is never permitted. Unless you explicitly enable anonymous access, all requests to a container and its blobs must be authorized. When you configure a container's public access level setting to permit anonymous access, clients can read data in that container without authorizing the request.

## WARNING

When a container is configured for public access, any client can read data in that container. Public access presents a potential security risk, so if your scenario does not require it, Microsoft recommends that you disallow it for the storage account. For more information, see [Prevent anonymous public read access to containers and blobs](#).

This article describes how to configure anonymous public read access for a container and its blobs. For information about how to access blob data anonymously from a client application, see [Access public containers and blobs anonymously with .NET](#).

## About anonymous public read access

Public access to your data is always prohibited by default. There are two separate settings that affect public access:

- Allow public access for the storage account.** By default, a storage account allows a user with the appropriate permissions to enable public access to a container. Blob data is not available for public access unless the user takes the additional step to explicitly configure the container's public access setting.
- Configure the container's public access setting.** By default, a container's public access setting is disabled, meaning that authorization is required for every request to the container or its data. A user with the appropriate permissions can modify a container's public access setting to enable anonymous access only if anonymous access is allowed for the storage account.

The following table summarizes how both settings together affect public access for a container.

	PUBLIC ACCESS LEVEL FOR THE CONTAINER IS SET TO PRIVATE (DEFAULT SETTING)	PUBLIC ACCESS LEVEL FOR THE CONTAINER IS SET TO CONTAINER	PUBLIC ACCESS LEVEL FOR THE CONTAINER IS SET TO BLOB
Public access is disallowed for the storage account	No public access to any container in the storage account.	No public access to any container in the storage account. The storage account setting overrides the container setting.	No public access to any container in the storage account. The storage account setting overrides the container setting.
Public access is allowed for the storage account (default setting)	No public access to this container (default configuration).	Public access is permitted to this container and its blobs.	Public access is permitted to blobs in this container, but not to the container itself.

When anonymous public access is permitted for a storage account and configured for a specific container, then a request to read a blob in that container that is passed without an *Authorization* header is accepted by the

service, and the blob's data is returned in the response.

## Allow or disallow public read access for a storage account

By default, a storage account is configured to allow a user with the appropriate permissions to enable public access to a container. When public access is allowed, a user with the appropriate permissions can modify a container's public access setting to enable anonymous public access to the data in that container. Blob data is never available for public access unless the user takes the additional step to explicitly configure the container's public access setting.

Keep in mind that public access to a container is always turned off by default and must be explicitly configured to permit anonymous requests. Regardless of the setting on the storage account, your data will never be available for public access unless a user with appropriate permissions takes this additional step to enable public access on the container.

Disallowing public access for the storage account prevents anonymous access to all containers and blobs in that account. When public access is disallowed for the account, it is not possible to configure the public access setting for a container to permit anonymous access. For improved security, Microsoft recommends that you disallow public access for your storage accounts unless your scenario requires that users access blob resources anonymously.

### IMPORTANT

Disallowing public access for a storage account overrides the public access settings for all containers in that storage account. When public access is disallowed for the storage account, any future anonymous requests to that account will fail. Before changing this setting, be sure to understand the impact on client applications that may be accessing data in your storage account anonymously. For more information, see [Prevent anonymous public read access to containers and blobs](#).

To allow or disallow public access for a storage account, configure the account's **AllowBlobPublicAccess** property. This property is available for all storage accounts that are created with the Azure Resource Manager deployment model. For more information, see [Storage account overview](#).

The **AllowBlobPublicAccess** property is not set for a storage account by default and does not return a value until you explicitly set it. The storage account permits public access when the property value is either **null** or **true**.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Template](#)

To allow or disallow public access for a storage account in the Azure portal, follow these steps:

1. Navigate to your storage account in the Azure portal.
2. Locate the **Configuration** setting under **Settings**.
3. Set **Blob public access** to **Enabled** or **Disabled**.



#### **NOTE**

Disallowing public access for a storage account does not affect any static websites hosted in that storage account. The **\$web** container is always publicly accessible.

After you update the public access setting for the storage account, it may take up to 30 seconds before the change is fully propagated.

When a container is configured for anonymous public access, requests to read blobs in that container do not need to be authorized. However, any firewall rules that are configured for the storage account remain in effect and will block anonymous traffic.

Allowing or disallowing blob public access requires version 2019-04-01 or later of the Azure Storage resource provider. For more information, see [Azure Storage Resource Provider REST API](#).

The examples in this section showed how to read the **AllowBlobPublicAccess** property for the storage account to determine if public access is currently allowed or disallowed. To learn more about how to verify that an account's public access setting is configured to prevent anonymous access, see [Remediate anonymous public access](#).

## Set the public access level for a container

To grant anonymous users read access to a container and its blobs, first allow public access for the storage account, then set the container's public access level. If public access is denied for the storage account, you will not be able to configure public access for a container.

When public access is allowed for a storage account, you can configure a container with the following permissions:

- **No public read access:** The container and its blobs can be accessed only with an authorized request. This option is the default for all new containers.
- **Public read access for blobs only:** Blobs within the container can be read by anonymous request, but container data is not available anonymously. Anonymous clients cannot enumerate the blobs within the container.
- **Public read access for container and its blobs:** Container and blob data can be read by anonymous request, except for container permission settings and container metadata. Clients can enumerate blobs within the container by anonymous request, but cannot enumerate containers within the storage account.

You cannot change the public access level for an individual blob. Public access level is set only at the container level. You can set the container's public access level when you create the container, or you can update the setting on an existing container.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Template](#)

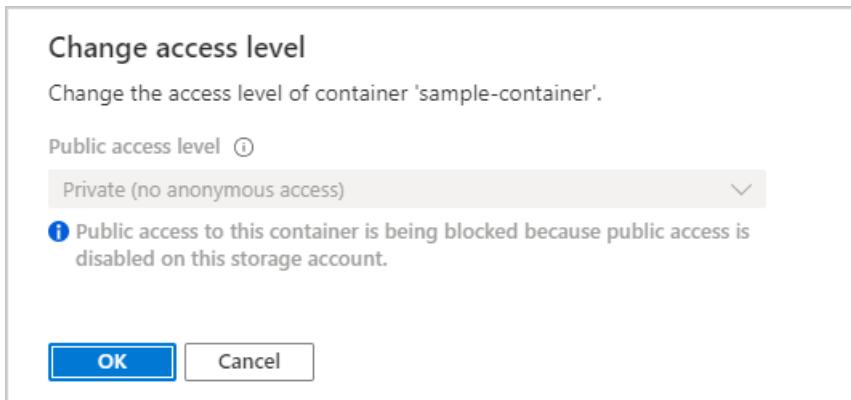
To update the public access level for one or more existing containers in the Azure portal, follow these steps:

1. Navigate to your storage account overview in the Azure portal.
2. Under **Data storage** on the menu blade, select **Blob containers**.
3. Select the containers for which you want to set the public access level.
4. Use the **Change access level** button to display the public access settings.

- Select the desired public access level from the **Public access level** dropdown and click the OK button to apply the change to the selected containers.

The screenshot shows the Azure portal interface for managing a storage account named 'exampleaccountnametest'. In the left sidebar, under 'Data storage', the 'Blob containers' option is selected and highlighted with a red box. On the right, the 'Change access level' dialog is open, also with a red box around its title. The 'Public access level' dropdown is expanded, showing three options: 'Private (no anonymous access)', 'Private (no anonymous access)', and 'Blob (anonymous read access for blobs only)'. Below the dropdown, two containers are listed: 'sample-container-4' and 'sample-container-5', both of which have the 'Private' access level applied. The 'Container' and 'Change access level' buttons at the top of the dialog are also highlighted with red boxes.

When public access is disallowed for the storage account, a container's public access level cannot be set. If you attempt to set the container's public access level, you'll see that the setting is disabled because public access is disallowed for the account.



## Check the public access setting for a set of containers

It is possible to check which containers in one or more storage accounts are configured for public access by listing the containers and checking the public access setting. This approach is a practical option when a storage account does not contain a large number of containers, or when you are checking the setting across a small number of storage accounts. However, performance may suffer if you attempt to enumerate a large number of containers.

The following example uses PowerShell to get the public access setting for all containers in a storage account. Remember to replace the placeholder values in brackets with your own values:

```
$rgName = "<resource-group>"
$accountName = "<storage-account>"

$storageAccount = Get-AzStorageAccount -ResourceGroupName $rgName -Name $accountName
$ctx = $storageAccount.Context

Get-AzStorageContainer -Context $ctx | Select Name, PublicAccess
```

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## Next steps

- [Prevent anonymous public read access to containers and blobs](#)
- [Access public containers and blobs anonymously with .NET](#)
- [Authorizing access to Azure Storage](#)

# Prevent anonymous public read access to containers and blobs

8/22/2022 • 15 minutes to read • [Edit Online](#)

Anonymous public read access to containers and blobs in Azure Storage is a convenient way to share data, but may also present a security risk. It's important to manage anonymous access judiciously and to understand how to evaluate anonymous access to your data. Operational complexity, human error, or malicious attack against data that is publicly accessible can result in costly data breaches. Microsoft recommends that you enable anonymous access only when necessary for your application scenario.

By default, public access to your blob data is always prohibited. However, the default configuration for a storage account permits a user with appropriate permissions to configure public access to containers and blobs in a storage account. For enhanced security, you can disallow all public access to storage account, regardless of the public access setting for an individual container. Disallowing public access to the storage account prevents a user from enabling public access for a container in the account. Microsoft recommends that you disallow public access to a storage account unless your scenario requires it. Disallowing public access helps to prevent data breaches caused by undesired anonymous access.

When you disallow public blob access for the storage account, Azure Storage rejects all anonymous requests to that account. After public access is disallowed for an account, containers in that account cannot be subsequently configured for public access. Any containers that have already been configured for public access will no longer accept anonymous requests. For more information, see [Configure anonymous public read access for containers and blobs](#).

This article describes how to use a DRAG (Detection-Remediation-Audit-Governance) framework to continuously manage public access for your storage accounts.

## Detect anonymous requests from client applications

When you disallow public read access for a storage account, you risk rejecting requests to containers and blobs that are currently configured for public access. Disallowing public access for a storage account overrides the public access settings for individual containers in that storage account. When public access is disallowed for the storage account, any future anonymous requests to that account will fail.

To understand how disallowing public access may affect client applications, Microsoft recommends that you enable logging and metrics for that account and analyze patterns of anonymous requests over an interval of time. Use metrics to determine the number of anonymous requests to the storage account, and use logs to determine which containers are being accessed anonymously.

### Monitor anonymous requests with Metrics Explorer

To track anonymous requests to a storage account, use Azure Metrics Explorer in the Azure portal. For more information about Metrics Explorer, see [Getting started with Azure Metrics Explorer](#).

Follow these steps to create a metric that tracks anonymous requests:

1. Navigate to your storage account in the Azure portal. Under the **Monitoring** section, select **Metrics**.
2. Select **Add metric**. In the **Metric** dialog, specify the following values:
  - a. Leave the **Scope** field set to the name of the storage account.
  - b. Set the **Metric Namespace** to *Blob*. This metric will report requests against Blob storage only.

c. Set the **Metric** field to *Transactions*.

d. Set the **Aggregation** field to *Sum*.

The new metric will display the sum of the number of transactions against Blob storage over a given interval of time. The resulting metric appears as shown in the following image:

3. Next, select the **Add filter** button to create a filter on the metric for anonymous requests.

4. In the **Filter** dialog, specify the following values:

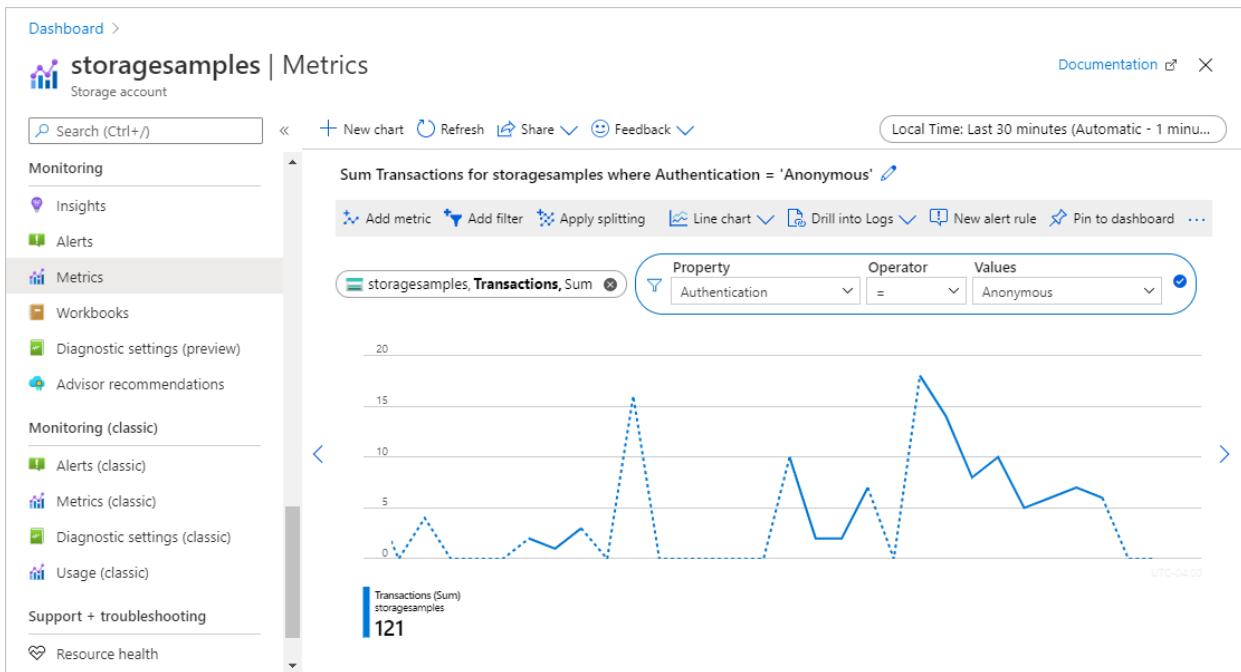
a. Set the **Property** value to *Authentication*.

b. Set the **Operator** field to the equal sign (=).

c. Set the **Values** field to *Anonymous* by selecting it from the dropdown or typing it in.

5. In the upper-right corner, select the time interval over which you want to view the metric. You can also indicate how granular the aggregation of requests should be, by specifying intervals anywhere from 1 minute to 1 month.

After you have configured the metric, anonymous requests will begin to appear on the graph. The following image shows anonymous requests aggregated over the past thirty minutes.



You can also configure an alert rule to notify you when a certain number of anonymous requests are made against your storage account. For more information, see [Create, view, and manage metric alerts using Azure Monitor](#).

### Analyze logs to identify containers receiving anonymous requests

Azure Storage logs capture details about requests made against the storage account, including how a request was authorized. You can analyze the logs to determine which containers are receiving anonymous requests.

To log requests to your Azure Storage account in order to evaluate anonymous requests, you can use Azure Storage logging in Azure Monitor (preview). For more information, see [Monitor Azure Storage](#).

Azure Storage logging in Azure Monitor supports using log queries to analyze log data. To query logs, you can use an Azure Log Analytics workspace. To learn more about log queries, see [Tutorial: Get started with Log Analytics queries](#).

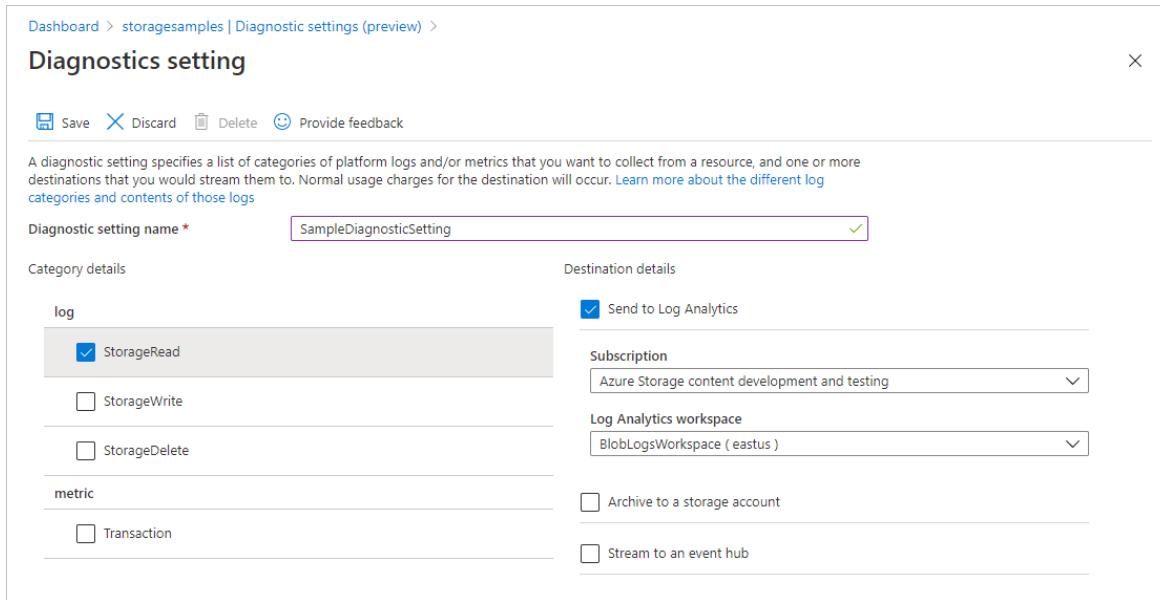
## NOTE

The preview of Azure Storage logging in Azure Monitor is supported only in the Azure public cloud. Government clouds do not support logging for Azure Storage with Azure Monitor.

### Create a diagnostic setting in the Azure portal

To log Azure Storage data with Azure Monitor and analyze it with Azure Log Analytics, you must first create a diagnostic setting that indicates what types of requests and for which storage services you want to log data. To create a diagnostic setting in the Azure portal, follow these steps:

1. Create a new Log Analytics workspace in the subscription that contains your Azure Storage account. After you configure logging for your storage account, the logs will be available in the Log Analytics workspace. For more information, see [Create a Log Analytics workspace in the Azure portal](#).
2. Navigate to your storage account in the Azure portal.
3. In the Monitoring section, select **Diagnostic settings (preview)**.
4. Select **Blob** to log requests made against Blob storage.
5. Select **Add diagnostic setting**.
6. Provide a name for the diagnostic setting.
7. Under **Category details**, in the **log** section, choose which types of requests to log. All anonymous requests will be read requests, so select **StorageRead** to capture anonymous requests.
8. Under **Destination details**, select **Send to Log Analytics**. Select your subscription and the Log Analytics workspace you created earlier, as shown in the following image.



After you create the diagnostic setting, requests to the storage account are subsequently logged according to that setting. For more information, see [Create diagnostic setting to collect resource logs and metrics in Azure](#).

For a reference of fields available in Azure Storage logs in Azure Monitor, see [Resource logs \(preview\)](#).

### Query logs for anonymous requests

Azure Storage logs in Azure Monitor include the type of authorization that was used to make a request to a storage account. In your log query, filter on the **AuthenticationType** property to view anonymous requests.

To retrieve logs for the last 7 days for anonymous requests against Blob storage, open your Log Analytics workspace. Next, paste the following query into a new log query and run it:

```
StorageBlobLogs
| where TimeGenerated > ago(7d) and AuthenticationType == "Anonymous"
| project TimeGenerated, AccountName, AuthenticationType, Uri
```

You can also configure an alert rule based on this query to notify you about anonymous requests. For more information, see [Create, view, and manage log alerts using Azure Monitor](#).

## Remediate anonymous public access

After you have evaluated anonymous requests to containers and blobs in your storage account, you can take action to limit or prevent public access. If some containers in your storage account may need to be available for public access, then you can configure the public access setting for each container in your storage account. This option provides the most granular control over public access. For more information, see [Set the public access level for a container](#).

For enhanced security, you can disallow public access for the whole storage account. The public access setting for a storage account overrides the individual settings for containers in that account. When you disallow public access for a storage account, any containers that are configured to permit public access are no longer accessible anonymously. For more information, see [Allow or disallow public read access for a storage account](#).

If your scenario requires that certain containers need to be available for public access, it may be advisable to move those containers and their blobs into storage accounts that are reserved for public access. You can then disallow public access for any other storage accounts.

### Verify that public access to a blob is not permitted

To verify that public access to a specific blob is disallowed, you can attempt to download the blob via its URL. If the download succeeds, then the blob is still publicly available. If the blob is not publicly accessible because public access has been disallowed for the storage account, then you will see an error message indicating that public access is not permitted on this storage account.

The following example shows how to use PowerShell to attempt to download a blob via its URL. Remember to replace the placeholder values in brackets with your own values:

```
$url = "<absolute-url-to-blob>"
$downloadTo = "<file-path-for-download>"
Invoke-WebRequest -Uri $url -OutFile $downloadTo -ErrorAction Stop
```

### Verify that modifying the container's public access setting is not permitted

To verify that a container's public access setting cannot be modified after public access is disallowed for the storage account, you can attempt to modify the setting. Changing the container's public access setting will fail if public access is disallowed for the storage account.

The following example shows how to use PowerShell to attempt to change a container's public access setting. Remember to replace the placeholder values in brackets with your own values:

```
$rgName = "<resource-group>"
$accountName = "<storage-account>"
$containerName = "<container-name>"

$storageAccount = Get-AzStorageAccount -ResourceGroupName $rgName -Name $accountName
$ctx = $storageAccount.Context

Set-AzStorageContainerAcl -Context $ctx -Container $containerName -Permission Blob
```

### Verify that creating a container with public access enabled is not permitted

If public access is disallowed for the storage account, then you will not be able to create a new container with public access enabled. To verify, you can attempt to create a container with public access enabled.

The following example shows how to use PowerShell to attempt to create a container with public access enabled. Remember to replace the placeholder values in brackets with your own values:

```
$rgName = "<resource-group>"
$accountName = "<storage-account>"
$containerName = "<container-name>"

$storageAccount = Get-AzStorageAccount -ResourceGroupName $rgName -Name $accountName
$ctx = $storageAccount.Context

New-AzStorageContainer -Name $containerName -Permission Blob -Context $ctx
```

### Check the public access setting for multiple accounts

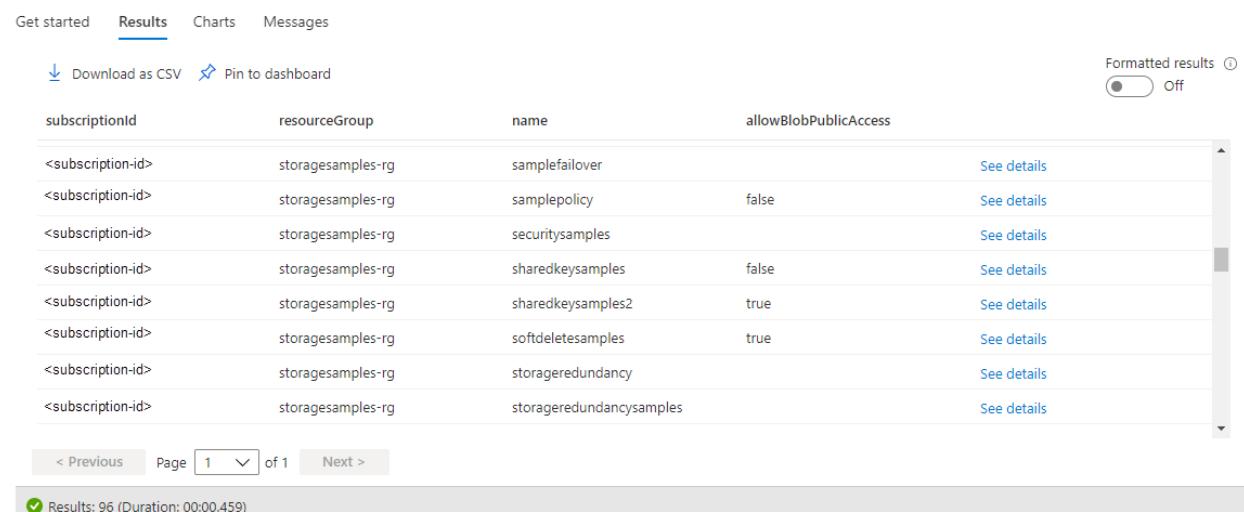
To check the public access setting across a set of storage accounts with optimal performance, you can use the Azure Resource Graph Explorer in the Azure portal. To learn more about using the Resource Graph Explorer, see [Quickstart: Run your first Resource Graph query using Azure Resource Graph Explorer](#).

The `AllowBlobPublicAccess` property is not set for a storage account by default and does not return a value until you explicitly set it. The storage account permits public access when the property value is either `null` or `true`.

Running the following query in the Resource Graph Explorer returns a list of storage accounts and displays public access setting for each account:

```
resources
| where type =~ 'Microsoft.Storage/storageAccounts'
| extend allowBlobPublicAccess = parse_json(properties).allowBlobPublicAccess
| project subscriptionId, resourceGroup, name, allowBlobPublicAccess
```

The following image shows the results of a query across a subscription. Note that for storage accounts where the `AllowBlobPublicAccess` property has been explicitly set, it appears in the results as `true` or `false`. If the `AllowBlobPublicAccess` property has not been set for a storage account, it appears as blank (or null) in the query results.



The screenshot shows the Azure Resource Graph Explorer interface. At the top, there are tabs for 'Get started', 'Results' (which is selected), 'Charts', and 'Messages'. Below the tabs are download and pinning options. On the right, there is a 'Formatted results' toggle switch set to 'Off'. The main area displays a table with the following columns: 'subscriptionId', 'resourceGroup', 'name', and 'allowBlobPublicAccess'. The table contains nine rows of data, each representing a storage account. The 'allowBlobPublicAccess' column shows values like 'false', 'true', and 'null' for different accounts. At the bottom of the table, there are navigation links for 'Previous', 'Page 1 of 1', and 'Next >'. A status bar at the bottom indicates 'Results: 96 (Duration: 00:00:459)'.

subscriptionId	resourceGroup	name	allowBlobPublicAccess
<subscription-id>	storagesamples-rg	samplefailover	See details
<subscription-id>	storagesamples-rg	samplepolicy	false
<subscription-id>	storagesamples-rg	securitysamples	See details
<subscription-id>	storagesamples-rg	sharedkeysamples	false
<subscription-id>	storagesamples-rg	sharedkeysamples2	true
<subscription-id>	storagesamples-rg	softdeletesamples	true
<subscription-id>	storagesamples-rg	storageredundancy	See details
<subscription-id>	storagesamples-rg	storageredundancysamples	See details

## Use Azure Policy to audit for compliance

If you have a large number of storage accounts, you may want to perform an audit to make sure that those accounts are configured to prevent public access. To audit a set of storage accounts for their compliance, use

Azure Policy. Azure Policy is a service that you can use to create, assign, and manage policies that apply rules to Azure resources. Azure Policy helps you to keep those resources compliant with your corporate standards and service level agreements. For more information, see [Overview of Azure Policy](#).

## Create a policy with an Audit effect

Azure Policy supports effects that determine what happens when a policy rule is evaluated against a resource. The Audit effect creates a warning when a resource is not in compliance, but does not stop the request. For more information about effects, see [Understand Azure Policy effects](#).

To create a policy with an Audit effect for the public access setting for a storage account with the Azure portal, follow these steps:

1. In the Azure portal, navigate to the Azure Policy service.
2. Under the **Authoring** section, select **Definitions**.
3. Select **Add policy definition** to create a new policy definition.
4. For the **Definition location** field, select the **More** button to specify where the audit policy resource is located.
5. Specify a name for the policy. You can optionally specify a description and category.
6. Under **Policy rule**, add the following policy definition to the **policyRule** section.

```
{
 "if": {
 "allOf": [
 {
 "field": "type",
 "equals": "Microsoft.Storage/storageAccounts"
 },
 {
 "not": {
 "field": "Microsoft.Storage/storageAccounts/allowBlobPublicAccess",
 "equals": "false"
 }
 }
]
 },
 "then": {
 "effect": "audit"
 }
}
```

7. Save the policy.

## Assign the policy

Next, assign the policy to a resource. The scope of the policy corresponds to that resource and any resources beneath it. For more information on policy assignment, see [Azure Policy assignment structure](#).

To assign the policy with the Azure portal, follow these steps:

1. In the Azure portal, navigate to the Azure Policy service.
2. Under the **Authoring** section, select **Assignments**.
3. Select **Assign policy** to create a new policy assignment.
4. For the **Scope** field, select the scope of the policy assignment.
5. For the **Policy definition** field, select the **More** button, then select the policy you defined in the previous section from the list.
6. Provide a name for the policy assignment. The description is optional.

7. Leave **Policy enforcement** set to *Enabled*. This setting has no effect on the audit policy.

8. Select **Review + create** to create the assignment.

## View compliance report

After you've assigned the policy, you can view the compliance report. The compliance report for an audit policy provides information on which storage accounts are not in compliance with the policy. For more information, see [Get policy compliance data](#).

It may take several minutes for the compliance report to become available after the policy assignment is created.

To view the compliance report in the Azure portal, follow these steps:

1. In the Azure portal, navigate to the Azure Policy service.
2. Select **Compliance**.
3. Filter the results for the name of the policy assignment that you created in the previous step. The report shows how many resources are not in compliance with the policy.
4. You can drill down into the report for additional details, including a list of storage accounts that are not in compliance.

The screenshot shows the 'Assignment Details' page in the Azure portal under the 'Policy compliance' section. The assignment is named 'Audit allow public blob access setting' and is assigned to the scope 'Azure Storage content development and testing/storagesamples-rg'. The 'Selected Scopes' dropdown shows '6 selected subscriptions'. The 'Compliance state' is 'Non-compliant' (indicated by a red 'X'). The 'Overall resource compliance' is '11%' (7 out of 63). The 'Non-compliant resources' count is '56' (out of 63). The 'Events (last 7 days)' section shows 0 Audit, 0 Append, 0 Modify, 0 Deny, and 0 Deploy events. The 'Resource compliance' tab is active, displaying a table of resources. The table columns include Name, Compliance state, Compliance reason, Resource type, Location, Scope, and Last evaluated. Five resources are listed as non-compliant: 'securitysamples', 'softdeletesamples', 'storageredundancy', 'storageredundancysamples', and 'storagesample'. Each row includes a 'Details' link and three vertical dots for more options.

## Use Azure Policy to enforce authorized access

Azure Policy supports cloud governance by ensuring that Azure resources adhere to requirements and standards. To ensure that storage accounts in your organization permit only authorized requests, you can create a policy that prevents the creation of a new storage account with a public access setting that allows anonymous requests. This policy will also prevent all configuration changes to an existing account if the public access setting for that account is not compliant with the policy.

The enforcement policy uses the Deny effect to prevent a request that would create or modify a storage account to allow public access. For more information about effects, see [Understand Azure Policy effects](#).

To create a policy with a Deny effect for a public access setting that allows anonymous requests, follow the same

steps described in [Use Azure Policy to audit for compliance](#), but provide the following JSON in the `policyRule` section of the policy definition:

```
{
 "if": {
 "allOf": [
 {
 "field": "type",
 "equals": "Microsoft.Storage/storageAccounts"
 },
 {
 "not": {
 "field": "Microsoft.Storage/storageAccounts/allowBlobPublicAccess",
 "equals": "false"
 }
 }
]
 },
 "then": {
 "effect": "deny"
 }
}
```

After you create the policy with the Deny effect and assign it to a scope, a user cannot create a storage account that allows public access. Nor can a user make any configuration changes to an existing storage account that currently allows public access. Attempting to do so results in an error. The public access setting for the storage account must be set to **false** to proceed with account creation or configuration.

The following image shows the error that occurs if you try to create a storage account that allows public access (the default for a new account) when a policy with a Deny effect requires that public access is disallowed.

**Errors** X

[Summary](#) [Raw Error](#)

**ERROR DETAILS** 

Resource 'testblobpublicaccess' was disallowed by policy. (Code: RequestDisallowedByPolicy)

Policy: [Deny blob public access](#)

---

WAS THIS HELPFUL?  

**Troubleshooting Options**

[Check Usage + Quota](#)   
[New Support Request](#) 

## Permissions for allowing or disallowing public access

To set the **AllowBlobPublicAccess** property for the storage account, a user must have permissions to create and manage storage accounts. Azure role-based access control (Azure RBAC) roles that provide these permissions include the **Microsoft.Storage/storageAccounts/write** or **Microsoft.Storage/storageAccounts/\*** action. Built-in roles with this action include:

- The Azure Resource Manager **Owner** role

- The Azure Resource Manager [Contributor](#) role
- The [Storage Account Contributor](#) role

These roles do not provide access to data in a storage account via Azure Active Directory (Azure AD). However, they include the **Microsoft.Storage/storageAccounts/listkeys/action**, which grants access to the account access keys. With this permission, a user can use the account access keys to access all data in a storage account.

Role assignments must be scoped to the level of the storage account or higher to permit a user to allow or disallow public access for the storage account. For more information about role scope, see [Understand scope for Azure RBAC](#).

Be careful to restrict assignment of these roles only to those who require the ability to create a storage account or update its properties. Use the principle of least privilege to ensure that users have the fewest permissions that they need to accomplish their tasks. For more information about managing access with Azure RBAC, see [Best practices for Azure RBAC](#).

#### NOTE

The classic subscription administrator roles Service Administrator and Co-Administrator include the equivalent of the Azure Resource Manager [Owner](#) role. The [Owner](#) role includes all actions, so a user with one of these administrative roles can also create and manage storage accounts. For more information, see [Classic subscription administrator roles, Azure roles, and Azure AD administrator roles](#).

## Next steps

- [Configure anonymous public read access for containers and blobs](#)
- [Access public containers and blobs anonymously with .NET](#)

# Access public containers and blobs anonymously with .NET

8/22/2022 • 2 minutes to read • [Edit Online](#)

Azure Storage supports optional public read access for containers and blobs. Clients can access public containers and blobs anonymously by using the Azure Storage client libraries, as well as by using other tools and utilities that support data access to Azure Storage.

This article shows how to access a public container or blob from .NET. For information about configuring anonymous read access on a container, see [Configure anonymous public read access for containers and blobs](#). For information about preventing all anonymous access to a storage account, see [Prevent anonymous public read access to containers and blobs](#).

A client that accesses containers and blobs anonymously can use constructors that do not require credentials. The following examples show a few different ways to reference containers and blobs anonymously.

## IMPORTANT

Any firewall rules that are in effect for the storage account apply even when public access is enabled for a container.

## Create an anonymous client object

You can create a new service client object for anonymous access by providing the Blob storage endpoint for the account. However, you must also know the name of a container in that account that's available for anonymous access.

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

```
public static void CreateAnonymousBlobClient()
{
 // Create the client object using the Blob storage endpoint for your account.
 BlobServiceClient blobServiceClient = new BlobServiceClient
 (new Uri(@"https://storagesamples.blob.core.windows.net/"));

 // Get a reference to a container that's available for anonymous access.
 BlobContainerClient container = blobServiceClient.GetBlobContainerClient("sample-container");

 // Read the container's properties.
 // Note this is only possible when the container supports full public read access.
 Console.WriteLine(container.GetProperties().Value.LastModified);
 Console.WriteLine(container.GetProperties().Value.ETag);
}
```

## Reference a container anonymously

If you have the URL to a container that is anonymously available, you can use it to reference the container directly.

- [.NET v12 SDK](#)

- [.NET v11 SDK](#)

```
public static void ListBlobsAnonymously()
{
 // Get a reference to a container that's available for anonymous access.
 BlobContainerClient container = new BlobContainerClient
 (new Uri(@"https://storagesamples.blob.core.windows.net/sample-container"));

 // List blobs in the container.
 // Note this is only possible when the container supports full public read access.
 foreach (BlobItem blobItem in container.GetBlobs())
 {
 Console.WriteLine(container.GetBlockBlobClient(blobItem.Name).Uri);
 }
}
```

## Reference a blob anonymously

If you have the URL to a blob that is available for anonymous access, you can reference the blob directly using that URL:

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

```
public static void DownloadBlobAnonymously()
{
 BlockBlobClient blob = new BlockBlobClient
 (new Uri(@"https://storagesamples.blob.core.windows.net/sample-container/logfile.txt"));
 blob.DownloadTo(@"C:\Temp\logfile.txt");
}
```

## Next steps

- [Configure anonymous public read access for containers and blobs](#)
- [Prevent anonymous public read access to containers and blobs](#)
- [Authorizing access to Azure Storage](#)

# Check the encryption status of a blob

8/22/2022 • 2 minutes to read • [Edit Online](#)

Every block blob, append blob, or page blob that was written to Azure Storage after October 20, 2017 is encrypted with Azure Storage encryption. Blobs created prior to this date continue to be encrypted by a background process.

This article shows how to determine whether a given blob has been encrypted.

## Check a blob's encryption status

Use the Azure portal, PowerShell, or Azure CLI to determine whether a blob is encrypted without code.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To use the Azure portal to check whether a blob has been encrypted, follow these steps:

1. In the Azure portal, navigate to your storage account.
2. Select **Containers** to navigate to a list of containers in the account.
3. Locate the blob and display its **Overview** tab.
4. View the **Server Encrypted** property. If **True**, as shown in the following image, then the blob is encrypted. Notice that the blob's properties also include the date and time that the blob was created.

The screenshot shows the Azure portal interface for a blob named "blob1.txt". The "Overview" tab is selected. The properties listed include:

Property	Value
URL	<a href="https://storagesamples.blob.core.windows.net/sampl...">https://storagesamples.blob.core.windows.net/sampl...</a>
LAST MODIFIED	10/15/2019, 3:51:58 PM
CREATION TIME	2/25/2019, 2:59:30 PM
TYPE	Block blob
SIZE	21 B
ACCESS TIER	Hot (Inferred)
ACCESS TIER LAST MODIFIED	N/A
SERVER ENCRYPTED	true
ETAG	0x8D751A923E07042
CONTENT-TYPE	application/octet-stream
CONTENT-MD5	m91jBTSTqgiwHi8TdHPing==
LEASE STATUS	Unlocked
LEASE STATE	Available
LEASE DURATION	-
COPY STATUS	-
COPY COMPLETION TIME	-

## Force encryption of a blob

If a blob that was created prior to October 20, 2017 has not yet been encrypted by the background process, you can force encryption to occur immediately by downloading and re-uploading the blob. A simple way to do this is with AzCopy.

To download a blob to your local file system with AzCopy, use the following syntax:

```
azcopy copy 'https://<storage-account-name>.<blob or dfs>.core.windows.net/<container-name>/<blob-path>' '<local-file-path>'
```

Example:

```
azcopy copy 'https://storagesamples.blob.core.windows.net/sample-container/blob1.txt' 'C:\temp\blob1.txt'
```

To re-upload the blob to Azure Storage with AzCopy, use the following syntax:

```
azcopy copy '<local-file-path>' 'https://<storage-account-name>.<blob or dfs>.core.windows.net/<container-name>/<blob-name>'
```

Example:

```
azcopy copy 'C:\temp\blob1.txt' 'https://storagesamples.blob.core.windows.net/sample-container/blob1.txt'
```

For more information about using AzCopy to copy blob data, see [Transfer data with AzCopy and Blob storage](#).

## Next steps

[Azure Storage encryption for data at rest](#)

# Determine which Azure Storage encryption key model is in use for the storage account

8/22/2022 • 2 minutes to read • [Edit Online](#)

Data in your storage account is automatically encrypted by Azure Storage. Azure Storage encryption offers two options for managing encryption keys at the level of the storage account:

- **Microsoft-managed keys.** By default, Microsoft manages the keys used to encrypt your storage account.
- **Customer-managed keys.** You can optionally choose to manage encryption keys for your storage account. Customer-managed keys must be stored in Azure Key Vault.

Additionally, you can provide an encryption key at the level of an individual request for some Blob storage operations. When an encryption key is specified on the request, that key overrides the encryption key that is active on the storage account. For more information, see [Specify a customer-provided key on a request to Blob storage](#).

For more information about encryption keys, see [Azure Storage encryption for data at rest](#).

## Check the encryption key model for the storage account

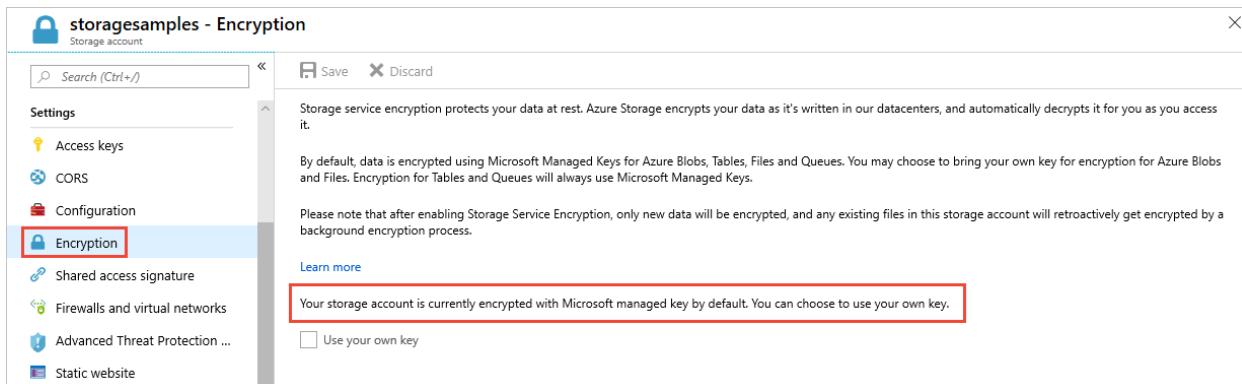
To determine whether a storage account is using Microsoft-managed keys or customer-managed keys for encryption, use one of the following approaches.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To check the encryption model for the storage account by using the Azure portal, follow these steps:

1. In the Azure portal, navigate to your storage account.
2. Select the **Encryption** setting and note the setting.

The following image shows a storage account that is encrypted with Microsoft-managed keys:



And the following image shows a storage account that is encrypted with customer-managed keys:

Dashboard > storagesamples - Encryption

**storagesamples - Encryption**  
Storage account

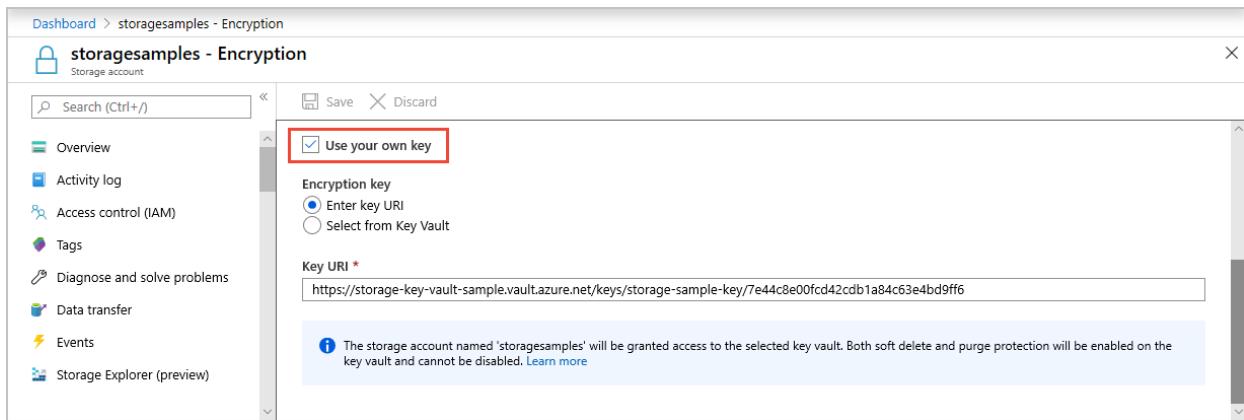
Search (Ctrl+ /) Save Discard

**Use your own key**

Encryption key  
 Enter key URI  
 Select from Key Vault

Key URI \*  
`https://storage-key-vault-sample.vault.azure.net/keys/storage-sample-key/7e44c8e0fc42cdb1a84c63e4bd9ff6`

The storage account named 'storagesamples' will be granted access to the selected key vault. Both soft delete and purge protection will be enabled on the key vault and cannot be disabled. [Learn more](#)



## Next steps

- [Azure Storage encryption for data at rest](#)
- [Customer-managed keys for Azure Storage encryption](#)

# Configure encryption with customer-managed keys stored in Azure Key Vault

8/22/2022 • 18 minutes to read • [Edit Online](#)

Azure Storage encrypts all data in a storage account at rest. By default, data is encrypted with Microsoft-managed keys. For additional control over encryption keys, you can manage your own keys. Customer-managed keys must be stored in Azure Key Vault or Key Vault Managed Hardware Security Model (HSM).

This article shows how to configure encryption with customer-managed keys stored in a key vault by using the Azure portal, PowerShell, or Azure CLI. To learn how to configure encryption with customer-managed keys stored in a managed HSM, see [Configure encryption with customer-managed keys stored in Azure Key Vault Managed HSM](#).

## NOTE

Azure Key Vault and Azure Key Vault Managed HSM support the same APIs and management interfaces for configuration.

## Configure a key vault

You can use a new or existing key vault to store customer-managed keys. The storage account and key vault may be in different regions or subscriptions in the same tenant. To learn more about Azure Key Vault, see [Azure Key Vault Overview](#) and [What is Azure Key Vault?](#).

Using customer-managed keys with Azure Storage encryption requires that both soft delete and purge protection be enabled for the key vault. Soft delete is enabled by default when you create a new key vault and cannot be disabled. You can enable purge protection either when you create the key vault or after it is created.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To learn how to create a key vault with the Azure portal, see [Quickstart: Create a key vault using the Azure portal](#). When you create the key vault, select **Enable purge protection**, as shown in the following image.

## Create key vault

X

Key vault name \* ⓘ storagekeysample ✓

Region \* West US ▾

Pricing tier \* ⓘ Standard ▾

Recovery options

Soft delete protection will automatically be enabled on this key vault. This feature allows you to recover or permanently delete a key vault and secrets for the duration of the retention period. This protection applies to the key vault and the secrets stored within the key vault.

To enforce a mandatory retention period and prevent the permanent deletion of key vaults or secrets prior to the retention period elapsing, you can turn on purge protection. When purge protection is enabled, secrets cannot be purged by users or by Microsoft.

Soft-delete ⓘ	Enabled
Days to retain deleted vaults * ⓘ	90
Purge protection ⓘ	<input type="radio"/> Disable purge protection (allow key vault and objects to be purged during retention period) <input checked="" type="radio"/> Enable purge protection (enforce a mandatory retention period for deleted vaults and vault objects)
<small>Once enabled, this option cannot be disabled</small>	

[Review + create](#)

&lt; Previous

Next : Access policy &gt;

To enable purge protection on an existing key vault, follow these steps:

1. Navigate to your key vault in the Azure portal.
2. Under **Settings**, choose **Properties**.
3. In the **Purge protection** section, choose **Enable purge protection**.

## Add a key

Next, add a key to the key vault.

Azure Storage encryption supports RSA and RSA-HSM keys of sizes 2048, 3072 and 4096. For more information about supported key types, see [About keys](#).

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To learn how to add a key with the Azure portal, see [Quickstart: Set and retrieve a key from Azure Key Vault using the Azure portal](#).

## Choose a managed identity to authorize access to the key vault

When you enable customer-managed keys for a storage account, you must specify a managed identity that will be used to authorize access to the key vault that contains the key. The managed identity must have permissions to access the key in the key vault.

The managed identity that authorizes access to the key vault may be either a user-assigned or system-assigned

managed identity, depending on your scenario:

- When you configure customer-managed keys at the time that you create a storage account, you must specify a user-assigned managed identity.
- When you configure customer-managed keys on an existing storage account, you can specify either a user-assigned managed identity or a system-assigned managed identity.

To learn more about system-assigned versus user-assigned managed identities, see [Managed identity types](#).

### Use a user-assigned managed identity to authorize access

A user-assigned is a standalone Azure resource. To learn how to create and manage a user-assigned managed identity, see [Manage user-assigned managed identities](#).

Both new and existing storage accounts can use a user-assigned identity to authorize access to the key vault. You must create the user-assigned identity before you configure customer-managed keys.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

When you configure customer-managed keys with the Azure portal, you can select an existing user-assigned identity through the portal user interface. For details, see one of the following sections:

- [Configure customer-managed keys for a new account](#)
- [Configure customer-managed keys for an existing account](#)

### Use a system-assigned managed identity to authorize access

A system-assigned managed identity is associated with an instance of an Azure service, in this case an Azure Storage account. You must explicitly assign a system-assigned managed identity to a storage account before you can use the system-assigned managed identity to authorize access to the key vault that contains your customer-managed key.

Only existing storage accounts can use a system-assigned identity to authorize access to the key vault. New storage accounts must use a user-assigned identity, if customer-managed keys are configured on account creation.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

When you configure customer-managed keys with the Azure portal with a system-assigned managed identity, the system-assigned managed identity is assigned to the storage account for you under the covers. For details, see [Configure customer-managed keys for an existing account](#).

## Configure the key vault access policy

The next step is to configure the key vault access policy. The key vault access policy grants permissions to the managed identity that will be used to authorize access to the key vault. To learn more about key vault access policies, see [Azure Key Vault Overview](#) and [Azure Key Vault security overview](#).

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To learn how to configure the key vault access policy with the Azure portal, see [Assign an Azure Key Vault access](#)

policy.

## Configure customer-managed keys for a new account

When you configure encryption with customer-managed keys for a new storage account, you can choose to automatically update the key version used for Azure Storage encryption whenever a new version is available in the associated key vault. Alternately, you can explicitly specify a key version to be used for encryption until the key version is manually updated.

You must use an existing user-assigned managed identity to authorize access to the key vault when you configure customer-managed keys while creating the storage account. The user-assigned managed identity must have appropriate permissions to access the key vault.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To configure customer-managed keys for a new storage account with automatic updating of the key version, follow these steps:

1. In the Azure portal, navigate to the **Storage accounts** page, and select the **Create** button to create a new account.
2. Follow the steps outlined in [Create a storage account](#) to fill out the fields on the **Basics**, **Advanced**, **Networking**, and **Data Protection** tabs.
3. On the **Encryption** tab, indicate for which services you want to enable support for customer-managed keys in the **Enable support for customer-managed keys** field.
4. In the **Encryption type** field, select **Customer-managed keys (CMK)**.
5. In the **Encryption key** field, choose **Select a key vault and key**, and specify the key vault and key.
6. For the **User-assigned identity** field, select an existing user-assigned managed identity.

## Create a storage account

Basics Advanced Networking Data protection **Encryption** Tags Review + create

Encryption type (i) \*

Microsoft-managed keys (MMK)

Customer-managed keys (CMK)

(i) This storage account will be granted access to the selected key vault. Both soft delete and purge protection will be enabled on the key vault and cannot be disabled.

Enable support for customer-managed keys (i)

Blobs and files only

All service types (blobs, files, tables, and queues)

⚠ This option cannot be changed after this storage account is created.

(i) In order to use customer-managed keys, the following resources will need to have been created beforehand. If they have not been created, you will need to leave this experience to go do so.

- A user-assigned identity that has Get, Wrap key, and Unwrap key permissions on the same key vault. [Learn more](#)

Encryption key \*

Select a key vault and key

Enter key from URI

Key vault and key \*

Key vault: storagekeysamples

Key: sample-key

Select a key vault and key

User-assigned identity (i) \*

sample-user-assigned-identity

Change

**Review + create**

< Previous

Next : Tags >

7. Select **Review + create** to validate and create the new account.

You can also configure customer-managed keys with manual updating of the key version when you create a new storage account. Follow the steps described in [Configure encryption for manual updating of key versions](#).

## Configure customer-managed keys for an existing account

When you configure encryption with customer-managed keys for an existing storage account, you can choose to automatically update the key version used for Azure Storage encryption whenever a new version is available in the associated key vault. Alternately, you can explicitly specify a key version to be used for encryption until the key version is manually updated.

You can use either a system-assigned or user-assigned managed identity to authorize access to the key vault when you configure customer-managed keys for an existing storage account.

### NOTE

To rotate a key, create a new version of the key in Azure Key Vault. Azure Storage does not handle key rotation, so you will need to manage rotation of the key in the key vault. You can [configure key auto-rotation in Azure Key Vault](#) or rotate your key manually.

## Configure encryption for automatic updating of key versions

Azure Storage can automatically update the customer-managed key that is used for encryption to use the latest key version from the key vault. Azure Storage checks the key vault daily for a new version of the key. When a new version becomes available, then Azure Storage automatically begins using the latest version of the key for encryption.

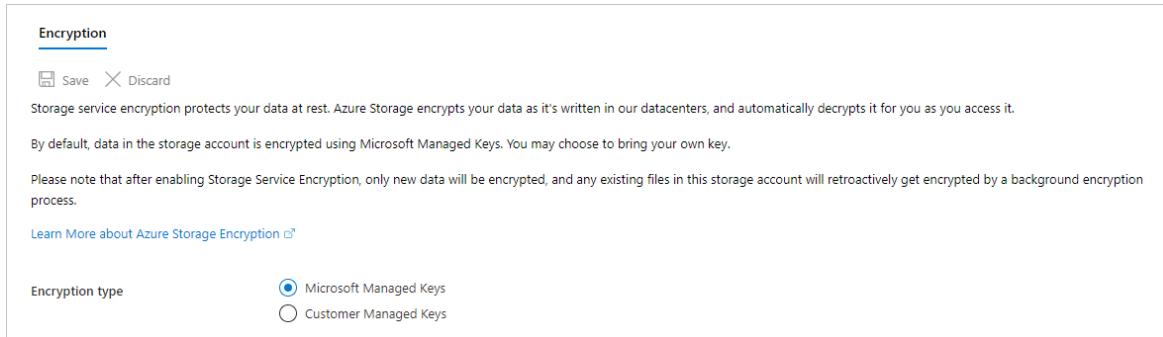
### IMPORTANT

Azure Storage checks the key vault for a new key version only once daily. When you rotate a key, be sure to wait 24 hours before disabling the older version.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To configure customer-managed keys for an existing account with automatic updating of the key version in the Azure portal, follow these steps:

1. Navigate to your storage account.
2. On the **Settings** blade for the storage account, click **Encryption**. By default, key management is set to **Microsoft Managed Keys**, as shown in the following image.



3. Select the **Customer Managed Keys** option.
4. Choose the **Select from Key Vault** option.
5. Select **Select a key vault and key**.
6. Select the key vault containing the key you want to use. You can also create a new vault.
7. Select the key from the key vault. You can also create a new key.

## Select a key

X

Subscription \*

Key store type  Key vault  Managed HSM

Key vault \*  [Create new key vault](#)

Key \*  [Create new key](#)

---

[Select](#) [Cancel](#)

8. Select the type of identity to use to authenticate access to the key vault. The options include **System-assigned** (the default) or **User-assigned**. To learn more about each type of managed identity, see [Managed identity types](#).

- If you select **System-assigned**, the system-assigned managed identity for the storage account is created under the covers, if it does not already exist.
- If you select **User-assigned**, then you must select an existing user-assigned identity that has permissions to access the key vault. To learn how to create a user-assigned identity, see [Manage user-assigned managed identities](#).

## Select user assigned manage...

X

Subscription \*

User assigned managed identities

 <b>sample-user-assigned-identity</b>	Resource Group: storagesamples-rg	
--------------------------------------------------------------------------------------------------------------------------	-----------------------------------	-------------------------------------------------------------------------------------

Selected identity:

No user assigned managed identity selected. Select a user assigned managed identity you want to assign to this resource.

[Add](#)

9. Save your changes.

After you've specified the key, the Azure portal indicates that automatic updating of the key version is enabled and displays the key version currently in use for encryption. The portal also displays the type of managed identity used to authorize access to the key vault and the principal ID for the managed identity.

## Encryption    Encryption scopes

Storage service encryption protects your data at rest. Azure Storage encrypts your data as it's written in our datacenters, and automatically decrypts it for you as you access it.

Please note that after enabling Storage Service Encryption, only new data will be encrypted, and any existing files in this storage account will retroactively get encrypted by a background encryption process. [Learn more about Azure Storage encryption](#)

### Encryption selection

Enable support for customer-managed keys	Blobs and files only
Infrastructure encryption	Disabled
Encryption type	<input type="radio"/> Microsoft-managed keys <input checked="" type="radio"/> Customer-managed keys
	<p><b>i</b> When customer-managed keys are enabled, the storage account named 'storagecmksamples1' is granted access to the selected key vault. Both soft delete and purge protection are also enabled on the key vault and cannot be disabled. <a href="#">Learn more</a></p>

### Key selection

Current key	<input type="text" value="https://storagekeysamples.vault.azure.net/keys/sample-key"/>
Automated key rotation	<input type="text" value="Enabled - Using the latest key version"/>
Key version in use	<input type="text" value="9e2a05c7d63a45d49ca2846407d5e98b"/>
Identity type	System-assigned
System-assigned identity	<input type="text" value="d7e0de46-fa63-42d9-b6a0-7f4da567f9d2"/>

[Change key](#)

[Save](#)

[Discard](#)

## Configure encryption for manual updating of key versions

If you prefer to manually update the key version, then explicitly specify the version at the time that you configure encryption with customer-managed keys. In this case, Azure Storage will not automatically update the key version when a new version is created in the key vault. To use a new key version, you must manually update the version used for Azure Storage encryption.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To configure customer-managed keys with manual updating of the key version in the Azure portal, specify the key URI, including the version. To specify a key as a URI, follow these steps:

1. To locate the key URI in the Azure portal, navigate to your key vault, and select the **Keys** setting. Select the desired key, then click the key to view its versions. Select a key version to view the settings for that version.
2. Copy the value of the **Key Identifier** field, which provides the URI.

Properties

Key Type RSA

RSA Key Size 2048

Created 4/9/2019, 12:50:38 PM

Updated 4/9/2019, 12:50:38 PM

Key Identifier

<key-uri>

Settings

Set activation date?

Set expiration date?

Enabled?  Yes  No

Tags

0 tags

Permitted operations

Encrypt  Verify  Import

Decrypt  Wrap Key

Sign  Unwrap Key

3. In the **Encryption key** settings for your storage account, choose the **Enter key URI** option.

4. Paste the URI that you copied into the **Key URI** field. Omit the key version from the URI to enable automatic updating of the key version.

Encryption [Encryption scopes](#)

Storage service encryption protects your data at rest. Azure Storage encrypts your data as it's written in our datacenters, and automatically decrypts it for you as you access it.

Please note that after enabling Storage Service Encryption, only new data will be encrypted, and any existing files in this storage account will retroactively get encrypted by a background encryption process. [Learn more about Azure Storage encryption](#)

Encryption selection

Enable support for customer-managed keys  Blobs and files only

Infrastructure encryption  Disabled

Encryption type  Microsoft-managed keys  Customer-managed keys

When customer-managed keys are enabled, the storage account named 'storagecmksamples1' is granted access to the selected key vault. Both soft delete and purge protection are also enabled on the key vault and cannot be disabled. [Learn more](#)

Key selection

Encryption key  Select from key vault  Enter key URI

Key URI ① <key-uri>

Subscription ① <subscription>

Identity type ①  System-assigned  User-assigned

**Save** **Discard**

5. Specify the subscription that contains the key vault.
6. Specify either a system-assigned or user-assigned managed identity.
7. Save your changes.

## Change the key

You can change the key that you are using for Azure Storage encryption at any time.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To change the key with the Azure portal, follow these steps:

1. Navigate to your storage account and display the **Encryption** settings.
2. Select the key vault and choose a new key.
3. Save your changes.

## Revoke customer-managed keys

Revoking a customer-managed key removes the association between the storage account and the key vault.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To revoke customer-managed keys with the Azure portal, disable the key as described in [Disable customer-managed keys](#).

## Disable customer-managed keys

When you disable customer-managed keys, your storage account is once again encrypted with Microsoft-managed keys.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To disable customer-managed keys in the Azure portal, follow these steps:

1. Navigate to your storage account and display the **Encryption** settings.
2. Deselect the checkbox next to the **Use your own key** setting.

## Next steps

- [Azure Storage encryption for data at rest](#)
- [Customer-managed keys for Azure Storage encryption](#)
- [Configure encryption with customer-managed keys stored in Azure Key Vault Managed HSM](#)

# Configure encryption with customer-managed keys stored in Azure Key Vault Managed HSM

8/22/2022 • 3 minutes to read • [Edit Online](#)

Azure Storage encrypts all data in a storage account at rest. By default, data is encrypted with Microsoft-managed keys. For additional control over encryption keys, you can manage your own keys. Customer-managed keys must be stored in Azure Key Vault or Key Vault Managed Hardware Security Model (HSM). An Azure Key Vault Managed HSM is an FIPS 140-2 Level 3 validated HSM.

This article shows how to configure encryption with customer-managed keys stored in a managed HSM by using Azure CLI. To learn how to configure encryption with customer-managed keys stored in a key vault, see [Configure encryption with customer-managed keys stored in Azure Key Vault](#).

## NOTE

Azure Key Vault and Azure Key Vault Managed HSM support the same APIs and management interfaces for configuration.

## Assign an identity to the storage account

First, assign a system-assigned managed identity to the storage account. You'll use this managed identity to grant the storage account permissions to access the managed HSM. For more information about system-assigned managed identities, see [What are managed identities for Azure resources?](#).

To assign a managed identity using Azure CLI, call [az storage account update](#). Remember to replace the placeholder values in brackets with your own values:

```
az storage account update \
--name <storage-account> \
--resource-group <resource_group> \
--assign-identity
```

## Assign a role to the storage account for access to the managed HSM

Next, assign the **Managed HSM Crypto Service Encryption User** role to the storage account's managed identity so that the storage account has permissions to the managed HSM. Microsoft recommends that you scope the role assignment to the level of the individual key in order to grant the fewest possible privileges to the managed identity.

To create the role assignment for storage account, call [az key vault role assignment create](#). Remember to replace the placeholder values in brackets with your own values.

```

storage_account_principal = $(az storage account show \
 --name <storage-account> \
 --resource-group <resource-group> \
 --query identity.principalId \
 --output tsv)

az keyvault role assignment create \
 --hsm-name <hsm-name> \
 --role "Managed HSM Crypto Service Encryption User" \
 --assignee $storage_account_principal \
 --scope /keys/<key-name>

```

## Configure encryption with a key in the managed HSM

Finally, configure Azure Storage encryption with customer-managed keys to use a key stored in the managed HSM. Supported key types include RSA-HSM keys of sizes 2048, 3072 and 4096. To learn how to create a key in a managed HSM, see [Create an HSM key](#).

Install Azure CLI 2.12.0 or later to configure encryption to use a customer-managed key in a managed HSM. For more information, see [Install the Azure CLI](#).

To automatically update the key version for a customer-managed key, omit the key version when you configure encryption with customer-managed keys for the storage account. For more information about configuring encryption for automatic key rotation, see [Update the key version](#).

Next, call [az storage account update](#) to update the storage account's encryption settings, as shown in the following example. Include the `--encryption-key-source` parameter and set it to `Microsoft.Keyvault` to enable customer-managed keys for the account. Remember to replace the placeholder values in brackets with your own values.

```

hsmurl = $(az keyvault show \
 --hsm-name <hsm-name> \
 --query properties.hsmUri \
 --output tsv)

az storage account update \
 --name <storage-account> \
 --resource-group <resource_group> \
 --encryption-key-name <key> \
 --encryption-key-source Microsoft.Keyvault \
 --encryption-key-vault $hsmurl

```

To manually update the version for a customer-managed key, include the key version when you configure encryption for the storage account:

```

az storage account update
 --name <storage-account> \
 --resource-group <resource_group> \
 --encryption-key-name <key> \
 --encryption-key-version $key_version \
 --encryption-key-source Microsoft.Keyvault \
 --encryption-key-vault $hsmurl

```

When you manually update the key version, you'll need to update the storage account's encryption settings to use the new version. First, query for the key vault URI by calling [az keyvault show](#), and for the key version by calling [az keyvault key list-versions](#). Then call [az storage account update](#) to update the storage account's encryption settings to use the new version of the key, as shown in the previous example.

## Next steps

- [Azure Storage encryption for data at rest](#)
- [Customer-managed keys for Azure Storage encryption](#)

# Specify a customer-provided key on a request to Blob storage with .NET

8/22/2022 • 2 minutes to read • [Edit Online](#)

Clients making requests against Azure Blob storage have the option to provide an AES-256 encryption key on an individual request. Including the encryption key on the request provides granular control over encryption settings for Blob storage operations. Customer-provided keys can be stored in Azure Key Vault or in another key store.

This article shows how to specify a customer-provided key on a request with .NET.

## Install client library packages

### NOTE

The examples shown here use the Azure Storage client library version 12. The version 12 client library is part of the Azure SDK. For more information about the Azure SDK, see the Azure SDK repository on [GitHub](#).

To install the Blob storage package, run the following command from the NuGet package manager console:

```
Install-Package Azure.Storage.Blobs
```

The examples shown here also use the latest version of the [Azure Identity client library for .NET](#) to authenticate with Azure AD credentials. To install the package, run the following command from the NuGet package manager console:

```
Install-Package Azure.Identity
```

To learn more about how to authenticate with the Azure Identity client library, see [Azure Identity client library for .NET](#).

## Use a customer-provided key to write to a blob

The following example provides an AES-256 key when uploading a blob with the v12 client library for Blob storage. The example uses the [DefaultAzureCredential](#) object to authorize the write request with Azure AD, but you can also authorize the request with Shared Key credentials. For more information about using the [DefaultAzureCredential](#) class to authorize a managed identity to access Azure Storage, see [Azure Identity client library for .NET](#).

```

async static Task UploadBlobWithClientKey(Uri blobUri,
 Stream data,
 byte[] key,
 string keySha256)
{
 // Create a new customer-provided key.
 // Key must be AES-256.
 var cpk = new CustomerProvidedKey(key);

 // Check the key's encryption hash.
 if (cpk.EncryptionKeyHash != keySha256)
 {
 throw new InvalidOperationException("The encryption key is corrupted.");
 }

 // Specify the customer-provided key on the options for the client.
 BlobClientOptions options = new BlobClientOptions()
 {
 CustomerProvidedKey = cpk
 };

 // Create the client object with options specified.
 BlobClient blobClient = new BlobClient(
 blobUri,
 new DefaultAzureCredential(),
 options);

 // If the container may not exist yet,
 // create a client object for the container.
 // The container client retains the credential and client options.
 BlobContainerClient containerClient =
 blobClient.GetParentBlobContainerClient();

 try
 {
 // Create the container if it does not exist.
 await containerClient.CreateIfNotExistsAsync();

 // Upload the data using the customer-provided key.
 await blobClient.UploadAsync(data);
 }
 catch (RequestFailedException e)
 {
 Console.WriteLine(e.Message);
 Console.ReadLine();
 throw;
 }
}

```

## Next steps

- [Provide an encryption key on a request to Blob storage](#)
- [Azure Storage encryption for data at rest](#)

# Create and manage encryption scopes

8/22/2022 • 14 minutes to read • [Edit Online](#)

Encryption scopes enable you to manage encryption at the level of an individual blob or container. You can use encryption scopes to create secure boundaries between data that resides in the same storage account but belongs to different customers. For more information about encryption scopes, see [Encryption scopes for Blob storage](#).

This article shows how to create an encryption scope. It also shows how to specify an encryption scope when you create a blob or container.

## Create an encryption scope

You can create an encryption scope that is protected with a Microsoft-managed key or with a customer-managed key that is stored in an Azure Key Vault or in an Azure Key Vault Managed Hardware Security Model (HSM). To create an encryption scope with a customer-managed key, you must first create a key vault or managed HSM and add the key you intend to use for the scope. The key vault or managed HSM must have purge protection enabled and must be in the same region as the storage account.

An encryption scope is automatically enabled when you create it. After you create the encryption scope, you can specify it when you create a blob. You can also specify a default encryption scope when you create a container, which automatically applies to all blobs in the container.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To create an encryption scope in the Azure portal, follow these steps:

1. Navigate to your storage account in the Azure portal.
2. Select the **Encryption** setting.
3. Select the **Encryption Scopes** tab.
4. Click the **Add** button to add a new encryption scope.
5. In the **Create Encryption Scope** pane, enter a name for the new scope.
6. Select the desired type of encryption key support, either **Microsoft-managed keys** or **Customer-managed keys**.
  - If you selected **Microsoft-managed keys**, click **Create** to create the encryption scope.
  - If you selected **Customer-managed keys**, then select a subscription and specify a key vault or a managed HSM and a key to use for this encryption scope.
7. If infrastructure encryption is enabled for the storage account, then it will automatically be enabled for the new encryption scope. Otherwise, you can choose whether to enable infrastructure encryption for the encryption scope.

## Create encryption scope

X

Encryption scope name \*

customer5scope



Encryption type

- Microsoft-managed keys
- Customer-managed keys

Encryption key

- Select from key vault
- Enter key URI

Subscription

Azure Storage content development and testing



Key vault \*

storageencryptionscopes



Key \*

customer5key



Infrastructure encryption (i)

- Disabled
- Enabled

**Create**

To learn how to configure Azure Storage encryption with customer-managed keys in a key vault or managed HSM, see the following articles:

- [Configure encryption with customer-managed keys stored in Azure Key Vault](#)
- [Configure encryption with customer-managed keys stored in Azure Key Vault Managed HSM](#)

To learn more about infrastructure encryption, see [Enable infrastructure encryption for double encryption of data](#).

## List encryption scopes for storage account

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To view the encryption scopes for a storage account in the Azure portal, navigate to the **Encryption Scopes** setting for the storage account. From this pane, you can enable or disable an encryption scope or change the key for an encryption scope.

Encryption scopes				
Actions		Details		
Name	Status	Encryption type	Key	Automated key rotation
customer1scope	Enabled	Microsoft-managed keys	-	-
customer2scope	Enabled	Customer-managed keys	customer2key	Enabled
customer3scope	Enabled	Customer-managed keys	customer3key	Disabled

To view details for a customer-managed key, including the key URI and version and whether the key version is automatically updated, follow the link in the **Key** column.

customer2key

Key URI  
https://storageencryptionscopes.vault.azure.net/keys/customer2key

Automated key rotation ⓘ  
Enabled

Key version in use ⓘ  
<key-version>

## Create a container with a default encryption scope

When you create a container, you can specify a default encryption scope. Blobs in that container will use that scope by default.

An individual blob can be created with its own encryption scope, unless the container is configured to require that all blobs use the default scope. For more information, see [Encryption scopes for containers and blobs](#).

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To create a container with a default encryption scope in the Azure portal, first create the encryption scope as described in [Create an encryption scope](#). Next, follow these steps to create the container:

1. Navigate to the list of containers in your storage account, and select the **Add** button to create a new container.
2. Expand the **Advanced** settings in the **New Container** pane.
3. In the **Encryption scope** drop-down, select the default encryption scope for the container.
4. To require that all blobs in the container use the default encryption scope, select the checkbox to **Use this encryption scope for all blobs in the container**. If this checkbox is selected, then an individual blob in the container cannot override the default encryption scope.

**New container**

Name \*

 ✓

Public access level ⓘ

Private (no anonymous access)

Advanced

Encryption scope

customer1

Use this encryption scope for all blobs in the container

**Create** **Discard**

The screenshot shows the 'New container' dialog in the Azure portal. It includes fields for 'Name' (set to 'container1'), 'Public access level' (set to 'Private (no anonymous access)'), and an 'Advanced' section for 'Encryption scope' (set to 'customer1'). A checkbox for 'Use this encryption scope for all blobs in the container' is checked. At the bottom are 'Create' and 'Discard' buttons.

If a client attempts to specify a scope when uploading a blob to a container that has a default encryption scope and the container is configured to prevent blobs from overriding the default scope, then the operation fails with a message indicating that the request is forbidden by the container encryption policy.

## Upload a blob with an encryption scope

When you upload a blob, you can specify an encryption scope for that blob, or use the default encryption scope for the container, if one has been specified.

When you upload a new blob with an encryption scope, you cannot change the default access tier for that blob.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To upload a blob with an encryption scope via the Azure portal, first create the encryption scope as described in [Create an encryption scope](#). Next, follow these steps to create the blob:

1. Navigate to the container to which you want to upload the blob.
2. Select the **Upload** button, and locate the blob to upload.
3. Expand the **Advanced** settings in the **Upload blob** pane.
4. Locate the **Encryption scope** drop-down section. By default, the blob is created with the default encryption scope for the container, if one has been specified. If the container requires that blobs use the default encryption scope, this section is disabled.
5. To specify a different scope for the blob that you are uploading, select **Choose an existing scope**, then select the desired scope from the drop-down.

## Upload blob

container2/

Files ⓘ

"helloworld.txt"

X

Overwrite if files already exist

Advanced

Authentication type ⓘ

Azure AD user account  Account key

Blob type ⓘ

Block blob

Upload .vhdx files as page blobs (recommended)

Block size ⓘ

4 MB

Access tier ⓘ

Hot (Inferred)

i Setting access tier is not supported with customer-key encryption.

Upload to folder

Encryption scope

- Use existing default container scope  
 Choose an existing scope

customer2

**Upload**

## Change the encryption key for a scope

To change the key that protects an encryption scope from a Microsoft-managed key to a customer-managed key, first make sure that you have enabled customer-managed keys with Azure Key Vault or Key Vault HSM for the storage account. For more information, see [Configure encryption with customer-managed keys stored in Azure Key Vault](#) or [Configure encryption with customer-managed keys stored in Azure Key Vault](#).

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To change the key that protects a scope in the Azure portal, follow these steps:

1. Navigate to the **Encryption Scopes** tab to view the list of encryption scopes for the storage account.
2. Select the **More** button next to the scope you wish to modify.
3. In the **Edit encryption scope** pane, you can change the encryption type from Microsoft-managed key to

customer-managed key or vice versa.

4. To select a new customer-managed key, select **Use a new key** and specify the key vault, key, and key version.

## Disable an encryption scope

When an encryption scope is disabled, you are no longer billed for it. Disable any encryption scopes that are not needed to avoid unnecessary charges. For more information, see [Azure Storage encryption for data at rest](#).

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To disable an encryption scope in the Azure portal, navigate to the **Encryption Scopes** setting for the storage account, select the desired encryption scope, and select **Disable**.

## Next steps

- [Azure Storage encryption for data at rest](#)
- [Encryption scopes for Blob storage](#)
- [Customer-managed keys for Azure Storage encryption](#)
- [Enable infrastructure encryption for double encryption of data](#)

# Enable infrastructure encryption for double encryption of data

8/22/2022 • 5 minutes to read • [Edit Online](#)

Azure Storage automatically encrypts all data in a storage account at the service level using 256-bit AES encryption, one of the strongest block ciphers available, and is FIPS 140-2 compliant. Customers who require higher levels of assurance that their data is secure can also enable 256-bit AES encryption at the Azure Storage infrastructure level for double encryption. Double encryption of Azure Storage data protects against a scenario where one of the encryption algorithms or keys may be compromised. In this scenario, the additional layer of encryption continues to protect your data.

Infrastructure encryption can be enabled for the entire storage account, or for an encryption scope within an account. When infrastructure encryption is enabled for a storage account or an encryption scope, data is encrypted twice — once at the service level and once at the infrastructure level — with two different encryption algorithms and two different keys.

Service-level encryption supports the use of either Microsoft-managed keys or customer-managed keys with Azure Key Vault or Key Vault Managed Hardware Security Model (HSM) (preview). Infrastructure-level encryption relies on Microsoft-managed keys and always uses a separate key. For more information about key management with Azure Storage encryption, see [About encryption key management](#).

To doubly encrypt your data, you must first create a storage account or an encryption scope that is configured for infrastructure encryption. This article describes how to enable infrastructure encryption.

## Create an account with infrastructure encryption enabled

To enable infrastructure encryption for a storage account, you must configure a storage account to use infrastructure encryption at the time that you create the account. Infrastructure encryption cannot be enabled or disabled after the account has been created. The storage account must be of type general-purpose v2 or premium block blob.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Template](#)

To use the Azure portal to create a storage account with infrastructure encryption enabled, follow these steps:

1. In the Azure portal, navigate to the **Storage accounts** page.
2. Choose the **Add** button to add a new general-purpose v2 or premium block blob storage account.
3. On the **Advanced** tab, locate **Infrastructure** encryption, and select **Enabled**.
4. Select **Review + create** to finish creating the storage account.

## Create storage account



Basics Networking Data protection Advanced Tags Review + create

### Security

Secure transfer required  Disabled  Enabled

Allow Blob public access  Disabled  Enabled

Allow shared key access  Disabled  Enabled

Minimum TLS version

Infrastructure encryption  Disabled  Enabled

**⚠ This option cannot be changed after this storage account is created.**

**Review + create**

< Previous

Next : Tags >

To verify that infrastructure encryption is enabled for a storage account with the Azure portal, follow these steps:

1. Navigate to your storage account in the Azure portal.

2. Under **Settings**, choose **Encryption**.

The screenshot shows the Azure Storage account settings for 'doubleencryptionsample'. The left sidebar lists various settings like Access keys, Geo-replication, CORS, Configuration, and Encryption. The Encryption section is currently selected and highlighted with a red border. The main content area displays information about storage service encryption and the configuration of infrastructure encryption. A secondary red border highlights the 'Infrastructure encryption' section, which shows the 'Enabled' radio button selected. Below it, the 'Encryption type' section shows 'Microsoft-managed keys' selected.

Encryption scopes (preview)

Save Discard

Storage service encryption protects your data at rest. Azure Storage encrypts your data as it's written in our datacenters, and automatically decrypts it for you as you access it.

By default, data in the storage account is encrypted using Microsoft-managed keys. You may choose to bring your own key.

Please note that after enabling Storage Service Encryption, only new data will be encrypted, and any existing files in this storage account will retroactively get encrypted by a background encryption process. [Learn more about Azure Storage encryption](#)

Infrastructure encryption  Disabled  Enabled

Encryption type  Microsoft-managed keys  Customer-managed keys

Azure Policy provides a built-in policy to require that infrastructure encryption be enabled for a storage account. For more information, see the [Storage](#) section in [Azure Policy built-in policy definitions](#).

## Create an encryption scope with infrastructure encryption enabled

If infrastructure encryption is enabled for an account, then any encryption scope created on that account automatically uses infrastructure encryption. If infrastructure encryption is not enabled at the account level, then you have the option to enable it for an encryption scope at the time that you create the scope. The infrastructure

encryption setting for an encryption scope cannot be changed after the scope is created. For more information, see [Create an encryption scope](#).

## Next steps

- [Azure Storage encryption for data at rest](#)
- [Customer-managed keys for Azure Storage encryption](#)
- [Encryption scopes for Blob storage](#)

# Client-side encryption for blobs

8/22/2022 • 12 minutes to read • [Edit Online](#)

The [Azure Blob Storage client library for .NET](#) supports encrypting data within client applications before uploading to Azure Storage, and decrypting data while downloading to the client. The library also supports integration with [Azure Key Vault](#) for storage account key management.

## IMPORTANT

Blob Storage supports both service-side and client-side encryption. For most scenarios, Microsoft recommends using service-side encryption features for ease of use in protecting your data. To learn more about service-side encryption, see [Azure Storage encryption for data at rest](#).

For a step-by-step tutorial that leads you through the process of encrypting blobs using client-side encryption and Azure Key Vault, see [Encrypt and decrypt blobs in Microsoft Azure Storage using Azure Key Vault](#).

## About client-side encryption

The Azure Blob Storage client library uses [AES](#) in order to encrypt user data. There are two versions of client-side encryption available in the client library:

- Version 2 uses [Galois/Counter Mode \(GCM\)](#) mode with AES.
- Version 1 uses [Cipher Block Chaining \(CBC\)](#) mode with AES.

## WARNING

Using version 1 of client-side encryption is no longer recommended due to a security vulnerability in the client library's implementation of CBC mode. For more information about this security vulnerability, see [Azure Storage updating client-side encryption in SDK to address security vulnerability](#). If you are currently using version 1, we recommend that you update your application to use version 2 and migrate your data. See the following section, [Mitigate the security vulnerability in your applications](#), for further guidance.

## Mitigate the security vulnerability in your applications

Due to a security vulnerability discovered in the Blob Storage client library's implementation of CBC mode, Microsoft recommends that you take one or more of the following actions immediately:

- Consider using service-side encryption features instead of client-side encryption. For more information about service-side encryption features, see [Azure Storage encryption for data at rest](#).
- If you need to use client-side encryption, then migrate your applications from client-side encryption v1 to client-side encryption v2.

The following table summarizes the steps you'll need to take if you choose to migrate your applications to client-side encryption v2:

CLIENT-SIDE ENCRYPTION STATUS	RECOMMENDED ACTIONS
-------------------------------	---------------------

CLIENT-SIDE ENCRYPTION STATUS	RECOMMENDED ACTIONS
Application is using client-side encryption a version of the client library that supports only client-side encryption v1.	<p>Update your application to use a version of the client library that supports client-side encryption v2. See <a href="#">SDK support matrix for client-side encryption</a> for a list of supported versions. <a href="#">Learn more...</a></p> <p>Update your code to use client-side encryption v2. <a href="#">Learn more...</a></p> <p>Download any encrypted data to decrypt it, then reencrypt it with client-side encryption v2. <a href="#">Learn more...</a></p>
Application is using client-side encryption with a version of the client library that supports client-side encryption v2.	<p>Update your code to use client-side encryption v2. <a href="#">Learn more...</a></p> <p>Download any encrypted data to decrypt it, then reencrypt it with client-side encryption v2. <a href="#">Learn more...</a></p>

Additionally, Microsoft recommends that you take the following steps to help secure your data:

- Configure your storage accounts to use private endpoints to secure all traffic between your virtual network (VNet) and your storage account over a private link. For more information, see [Use private endpoints for Azure Storage](#).
- Limit network access to specific networks only.

#### SDK support matrix for client-side encryption

The following table shows which versions of the client libraries for .NET, Java, and Python support which versions of client-side encryption:

	.NET	JAVA	PYTHON
Client-side encryption v2 and v1	Versions 12.13.0 and later	Versions 12.18.0 and later	Versions 12.13.0 and later
Client-side encryption v1 only	Versions 12.12.0 and earlier	Versions 12.17.0 and earlier	Versions 12.12.0 and earlier

If your application is using client-side encryption with an earlier version of the .NET, Java, or Python client library, you must first upgrade your code to a version that supports client-side encryption v2. Next, you must decrypt and re-encrypt your data with client-side encryption v2. If necessary, you can use a version of the client library that supports client-side encryption v2 side-by-side with an earlier version of the client library while you are migrating your code. For code examples, see [Example: Encrypting and decrypting a blob with client-side encryption v2](#).

## How client-side encryption works

The Azure Blob Storage client libraries use envelope encryption to encrypt and decrypt your data on the client side. Envelope encryption encrypts a key with one or more additional keys.

The Blob Storage client libraries rely on Azure Key Vault to protect the keys that are used for client-side encryption. For more information about Azure Key Vault, see [What is Azure Key Vault?](#).

#### Encryption and decryption via the envelope technique

Encryption via the envelope technique works as follows:

1. The Azure Storage client library generates a content encryption key (CEK), which is a one-time-use

symmetric key.

2. User data is encrypted using the CEK.
3. The CEK is then wrapped (encrypted) using the key encryption key (KEK). The KEK is identified by a key identifier and can be either an asymmetric key pair or a symmetric key. You can manage the KEK locally or store it in an Azure Key Vault.

The Azure Storage client library itself never has access to KEK. The library invokes the key wrapping algorithm that is provided by Key Vault. Users can choose to use custom providers for key wrapping/unwrapping if desired.

4. The encrypted data is then uploaded to Azure Blob Storage. The wrapped key together with some additional encryption metadata is stored as metadata on the blob.

Decryption via the envelope technique works as follows:

1. The Azure Storage client library assumes that the user is managing the KEK either locally or in an Azure Key Vault. The user doesn't need to know the specific key that was used for encryption. Instead, a key resolver that resolves different key identifiers to keys can be set up and used.
2. The client library downloads the encrypted data along with any encryption material that is stored in Azure Storage.
3. The wrapped CEK is then unwrapped (decrypted) using the KEK. The client library doesn't have access to the KEK during this process, but only invokes the unwrapping algorithm of the Azure Key Vault or other key store.
4. The client library uses the CEK to decrypt the encrypted user data.

### Encryption/decryption on blob upload/download

The Blob Storage client library supports encryption of whole blobs only on upload. For downloads, both complete and range downloads are supported.

During encryption, the client library generates a random initialization vector (IV) of 16 bytes and a random CEK of 32 bytes, and performs envelope encryption of the blob data using this information. The wrapped CEK and some additional encryption metadata are then stored as blob metadata along with the encrypted blob.

When a client downloads an entire blob, the wrapped CEK is unwrapped and used together with the IV to return the decrypted data to the client.

Downloading an arbitrary range in the encrypted blob involves adjusting the range provided by users in order to get a small amount of additional data that can be used to successfully decrypt the requested range.

All blob types (block blobs, page blobs, and append blobs) can be encrypted/decrypted using this scheme.

#### WARNING

If you are editing or uploading your own metadata for the blob, you must ensure that the encryption metadata is preserved. If you upload new metadata without also preserving the encryption metadata, then the wrapped CEK, IV, and other metadata will be lost and you will not be able to retrieve the contents of the blob. Calling the [Set Blob Metadata](#) operation always replaces all blob metadata.

When reading from or writing to an encrypted blob, use whole blob upload commands, such as [Put Blob](#), and range or whole blob download commands, such as [Get Blob](#). Avoid writing to an encrypted blob using protocol operations such as [Put Block](#), [Put Block List](#), [Put Page](#), or [Append Block](#). Calling these operations on an encrypted blob can corrupt it and make it unreadable.

## Example: Encrypting and decrypting a blob with client-side encryption

## v2

The code example in this section shows how to use client-side encryption v2 to encrypt and decrypt a blob.

### IMPORTANT

If you have data that has been previously encrypted with client-side encryption v1, then you'll need to decrypt that data and reencrypt it with client-side encryption v2. See the guidance and sample for your client library below.

- [.NET](#)
- [Java](#)
- [Python v12 SDK](#)

To use client-side encryption from your .NET code, reference the [Blob Storage client library](#). Make sure that you are using version 12.13.0 or later. If you need to migrate from version 11.x to version 12.13.0, see the [Migration guide](#).

Two additional packages are required for Azure Key Vault integration for client-side encryption:

- The [Azure.Core](#) package provides the `IKeyEncryptionKey` and `IKeyEncryptionKeyResolver` interfaces. The Blob Storage client library for .NET already defines this assembly as a dependency.
- The [Azure.Security.KeyVault.Keys](#) package (version 4.x and later) provides the Key Vault REST client and the cryptographic clients that are used with client-side encryption. You'll need to ensure that this package is referenced in your project if you're using Azure Key Vault as your key store.

Azure Key Vault is designed for high-value master keys, and throttling limits per key vault reflect this design. As of version 4.1.0 of [Azure.Security.KeyVault.Keys](#), the `IKeyEncryptionKeyResolver` interface doesn't support key caching. Should caching be necessary due to throttling, you can use the approach demonstrated in [this sample](#) to inject a caching layer into an `Azure.Security.KeyVault.Keys.Cryptography.KeyResolver` instance.

Developers can provide a key, a key resolver, or both a key and a key resolver. Keys are identified using a key identifier that provides the logic for wrapping and unwrapping the CEK. A key resolver is used to resolve a key during the decryption process. The key resolver defines a `resolve` method that returns a key given a key identifier. The resolver provides users the ability to choose between multiple keys that are managed in multiple locations.

On encryption, the key is used always and the absence of a key will result in an error.

On decryption, if the key is specified and its identifier matches the required key identifier, that key is used for decryption. Otherwise, the client library attempts to call the resolver. If there's no resolver specified, then the client library throws an error. If a resolver is specified, then the key resolver is invoked to get the key. If the resolver is specified but doesn't have a mapping for the key identifier, then the client library throws an error.

To use client-side encryption, create a `ClientSideEncryptionOptions` object and set it on client creation with `SpecializedBlobClientOptions`. You can't set encryption options on a per-API basis. Everything else will be handled by the client library internally.

```

// Your key and key resolver instances, either through Azure Key Vault SDK or an external implementation.
IKeyEncryptionKey key;
IKeyEncryptionKeyResolver keyResolver;

// Create the encryption options to be used for upload and download.
ClientSideEncryptionOptions encryptionOptions = new
ClientSideEncryptionOptions(ClientSideEncryptionVersion.V2_0)
{
 KeyEncryptionKey = key,
 KeyResolver = keyResolver,
 // String value that the client library will use when calling IKeyEncryptionKey.WrapKey()
 KeyWrapAlgorithm = "some algorithm name"
};

// Set the encryption options on the client options.
BlobClientOptions options = new SpecializedBlobClientOptions() { ClientSideEncryption = encryptionOptions };

// Create blob client with client-side encryption enabled.
// Client-side encryption options are passed from service clients to container clients,
// and from container clients to blob clients.
// Attempting to construct a BlockBlobClient, PageBlobClient, or AppendBlobClient from a BlobContainerClient
// with client-side encryption options present will throw, as this functionality is only supported with
// BlobClient.
BlobClient blob = new BlobServiceClient(connectionString, options).GetBlobContainerClient("my-
container").GetBlobClient("myBlob");

// Upload the encrypted contents to the blob.
blob.Upload(stream);

// Download and decrypt the encrypted contents from the blob.
MemoryStream outputStream = new MemoryStream();
blob.DownloadTo(outputStream);

```

You can apply encryption options to a **BlobServiceClient**, **BlobContainerClient**, or **BlobClient** constructors that accept **BlobClientOptions** objects.

If a **BlobClient** object already exists in your code but lacks client-side encryption options, then you can use an extension method to create a copy of that object with the given **ClientSideEncryptionOptions**. This extension method avoids the overhead of constructing a new **BlobClient** object from scratch.

```

using Azure.Storage.Blobs.Specialized;

// An existing BlobClient instance and encryption options.
BlobClient plaintextBlob;
ClientSideEncryptionOptions encryptionOptions;

// Get a copy of the blob that uses client-side encryption.
BlobClient clientSideEncryptionBlob = plaintextBlob.WithClientSideEncryptionOptions(encryptionOptions);

```

After you update your code to use client-side encryption v2, make sure that you deencrypt and reencrypt any existing encrypted data, as described in [Reencrypt previously encrypted data with client-side encryption v2](#).

## Reencrypt previously encrypted data with client-side encryption v2

Any data that was previously encrypted with client-side encryption v1 must be decrypted and then reencrypted with client-side encryption v2 to mitigate the security vulnerability. Decryption requires downloading the data and reencryption requires reuploading it to Blob Storage.

- [.NET](#)
- [Java](#)

- [Python](#)

For a sample project that shows how to migrate data from client-side encryption v1 to v2 and how to encrypt data with client-side encryption v2 in .NET, see the [Encryption migration sample project](#).

## Client-side encryption and performance

Keep in mind that encrypting your storage data results in additional performance overhead. When you use client-side encryption in your application, the client library must securely generate the CEK and IV, encrypt the content itself, communicate with your chosen keystore for key-enveloping, and format and upload additional metadata. This overhead varies depending on the quantity of data being encrypted. We recommend that customers always test their applications for performance during development.

## Next steps

- [Azure Storage updating client-side encryption in SDK to address security vulnerability](#)
- [Azure Storage encryption for data at rest](#)
- [Azure Key Vault documentation](#)

# Require secure transfer to ensure secure connections

8/22/2022 • 3 minutes to read • [Edit Online](#)

You can configure your storage account to accept requests from secure connections only by setting the **Secure transfer required** property for the storage account. When you require secure transfer, any requests originating from an insecure connection are rejected. Microsoft recommends that you always require secure transfer for all of your storage accounts.

When secure transfer is required, a call to an Azure Storage REST API operation must be made over HTTPS. Any request made over HTTP is rejected. By default, the **Secure transfer required** property is enabled when you create a storage account.

Azure Policy provides a built-in policy to ensure that secure transfer is required for your storage accounts. For more information, see the **Storage** section in [Azure Policy built-in policy definitions](#).

Connecting to an Azure file share over SMB without encryption fails when secure transfer is required for the storage account. Examples of insecure connections include those made over SMB 2.1 or SMB 3.x without encryption.

## NOTE

Because Azure Storage doesn't support HTTPS for custom domain names, this option is not applied when you're using a custom domain name.

This secure transfer setting does not apply to TCP. Connections via NFS 3.0 protocol support in Azure Blob Storage using TCP, which is not secured, will succeed.

## Require secure transfer in the Azure portal

You can turn on the **Secure transfer required** property when you create a storage account in the [Azure portal](#). You can also enable it for existing storage accounts.

### Require secure transfer for a new storage account

1. Open the **Create storage account** pane in the Azure portal.
2. In the **Advanced** page, select the **Enable secure transfer** checkbox.

## Create a storage account ...

X

Basics Advanced Networking Data protection Tags Review + create

### Security

Configure security settings that impact your storage account.

Enable secure transfer

Enable infrastructure encryption

Sign up is currently required to enable infrastructure encryption on a per-subscription basis. [Sign up](#)

Enable blob public access

Enable storage account key access

Minimum TLS version

[Review + create](#)

[< Previous](#)

[Next : Networking >](#)

### Require secure transfer for an existing storage account

1. Select an existing storage account in the Azure portal.
2. In the storage account menu pane, under **Settings**, select **Configuration**.
3. Under **Secure transfer required**, select **Enabled**.

The screenshot shows the Azure Storage Configuration page for the 'contoso' storage account. The top navigation bar includes 'Microsoft Azure', a search bar, and a 'Home > contoso' breadcrumb. On the left, a sidebar lists 'Settings' (selected), 'Configuration', 'Resource sharing (CORS)', 'Advisor recommendations', 'Endpoints', and 'Locks'. The main content area displays configuration options: 'Secure transfer required' (radio button set to 'Enabled'), 'Allow Blob public access' (radio button set to 'Enabled'), and 'Allow storage account key access' (radio button set to 'Enabled').

### Require secure transfer from code

To require secure transfer programmatically, set the `enableHttpsTrafficOnly` property to `True` on the storage account. You can set this property by using the Storage Resource Provider REST API, client libraries, or tools:

- [REST API](#)
- [PowerShell](#)
- [CLI](#)
- [NodeJS](#)
- [.NET SDK](#)
- [Python SDK](#)
- [Ruby SDK](#)

# Require secure transfer with PowerShell

## NOTE

This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

This sample requires the Azure PowerShell module Az version 0.7 or later. Run `Get-Module -ListAvailable Az` to find the version. If you need to install or upgrade, see [Install Azure PowerShell module](#).

Run `Connect-AzAccount` to create a connection with Azure.

Use the following command line to check the setting:

```
Get-AzStorageAccount -Name "{StorageAccountName}" -ResourceGroupName "{ResourceGroupName}"
StorageAccountName : {StorageAccountName}
Kind : Storage
EnableHttpsTrafficOnly : False
...
```

Use the following command line to enable the setting:

```
Set-AzStorageAccount -Name "{StorageAccountName}" -ResourceGroupName "{ResourceGroupName}" -
EnableHttpsTrafficOnly $True
StorageAccountName : {StorageAccountName}
Kind : Storage
EnableHttpsTrafficOnly : True
...
```

# Require secure transfer with Azure CLI

To run this sample, install the latest version of the [Azure CLI](#). To start, run `az login` to create a connection with Azure.

Samples for the Azure CLI are written for the `bash` shell. To run this sample in Windows PowerShell or Command Prompt, you may need to change elements of the script.

If you don't have an [Azure subscription](#), create an [Azure free account](#) before you begin.

Use the following command to check the setting:

```
az storage account show -g {ResourceGroupName} -n {StorageAccountName}
{
 "name": "{StorageAccountName}",
 "enableHttpsTrafficOnly": false,
 "type": "Microsoft.Storage/storageAccounts"
...
}
```

Use the following command to enable the setting:

```
az storage account update -g {ResourceGroupName} -n {StorageAccountName} --https-only true
{
 "name": "{StorageAccountName}",
 "enableHttpsTrafficOnly": true,
 "type": "Microsoft.Storage/storageAccounts"
 ...
}
```

## Next steps

[Security recommendations for Blob storage](#)

# Configure Azure Storage firewalls and virtual networks

8/22/2022 • 26 minutes to read • [Edit Online](#)

Azure Storage provides a layered security model. This model enables you to secure and control the level of access to your storage accounts that your applications and enterprise environments demand, based on the type and subset of networks or resources used. When network rules are configured, only applications requesting data over the specified set of networks or through the specified set of Azure resources can access a storage account. You can limit access to your storage account to requests originating from specified IP addresses, IP ranges, subnets in an Azure Virtual Network (VNet), or resource instances of some Azure services.

Storage accounts have a public endpoint that is accessible through the internet. You can also create [Private Endpoints for your storage account](#), which assigns a private IP address from your VNet to the storage account, and secures all traffic between your VNet and the storage account over a private link. The Azure storage firewall provides access control for the public endpoint of your storage account. You can also use the firewall to block all access through the public endpoint when using private endpoints. Your storage firewall configuration also enables select trusted Azure platform services to access the storage account securely.

An application that accesses a storage account when network rules are in effect still requires proper authorization for the request. Authorization is supported with Azure Active Directory (Azure AD) credentials for blobs and queues, with a valid account access key, or with an SAS token. When a blob container is configured for anonymous public access, requests to read data in that container do not need to be authorized, but the firewall rules remain in effect and will block anonymous traffic.

## IMPORTANT

Turning on firewall rules for your storage account blocks incoming requests for data by default, unless the requests originate from a service operating within an Azure Virtual Network (VNet) or from allowed public IP addresses. Requests that are blocked include those from other Azure services, from the Azure portal, from logging and metrics services, and so on.

You can grant access to Azure services that operate from within a VNet by allowing traffic from the subnet hosting the service instance. You can also enable a limited number of scenarios through the exceptions mechanism described below. To access data from the storage account through the Azure portal, you would need to be on a machine within the trusted boundary (either IP or VNet) that you set up.

## NOTE

This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

## Scenarios

To secure your storage account, you should first configure a rule to deny access to traffic from all networks (including internet traffic) on the public endpoint, by default. Then, you should configure rules that grant access to traffic from specific VNets. You can also configure rules to grant access to traffic from selected public internet IP address ranges, enabling connections from specific internet or on-premises clients. This configuration enables you to build a secure network boundary for your applications.

You can combine firewall rules that allow access from specific virtual networks and from public IP address ranges on the same storage account. Storage firewall rules can be applied to existing storage accounts, or when creating new storage accounts.

Storage firewall rules apply to the public endpoint of a storage account. You don't need any firewall access rules to allow traffic for private endpoints of a storage account. The process of approving the creation of a private endpoint grants implicit access to traffic from the subnet that hosts the private endpoint.

Network rules are enforced on all network protocols for Azure storage, including REST and SMB. To access data using tools such as the Azure portal, Storage Explorer, and AzCopy, explicit network rules must be configured.

Once network rules are applied, they're enforced for all requests. SAS tokens that grant access to a specific IP address serve to limit the access of the token holder, but don't grant new access beyond configured network rules.

Virtual machine disk traffic (including mount and unmount operations, and disk IO) is not affected by network rules. REST access to page blobs is protected by network rules.

Classic storage accounts do not support firewalls and virtual networks.

You can use unmanaged disks in storage accounts with network rules applied to back up and restore VMs by creating an exception. This process is documented in the [Manage Exceptions](#) section of this article. Firewall exceptions aren't applicable with managed disks as they're already managed by Azure.

## Change the default network access rule

By default, storage accounts accept connections from clients on any network. You can limit access to selected networks **or** prevent traffic from all networks and permit access only through a [private endpoint](#).

### WARNING

Changing this setting can impact your application's ability to connect to Azure Storage. Make sure to grant access to any allowed networks or set up access through a [private endpoint](#) before you change this setting.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

1. Go to the storage account you want to secure.
2. Locate the **Networking** settings under **Security + networking**.
3. Choose which type of public network access you want to allow.
  - To allow traffic from all networks, select **Enabled from all networks**.
  - To allow traffic only from specific virtual networks, select **Enabled from selected virtual networks and IP addresses**.
  - To block traffic from all networks, select **Disabled**.
4. Select **Save** to apply your changes.

## Grant access from a virtual network

You can configure storage accounts to allow access only from specific subnets. The allowed subnets may belong to a VNet in the same subscription, or those in a different subscription, including subscriptions belonging to a

different Azure Active Directory tenant.

You can enable a [Service endpoint](#) for Azure Storage within the VNet. The service endpoint routes traffic from the VNet through an optimal path to the Azure Storage service. The identities of the subnet and the virtual network are also transmitted with each request. Administrators can then configure network rules for the storage account that allow requests to be received from specific subnets in a VNet. Clients granted access via these network rules must continue to meet the authorization requirements of the storage account to access the data.

Each storage account supports up to 200 virtual network rules, which may be combined with [IP network rules](#).

#### IMPORTANT

If you delete a subnet that has been included in a network rule, it will be removed from the network rules for the storage account. If you create a new subnet by the same name, it will not have access to the storage account. To allow access, you must explicitly authorize the new subnet in the network rules for the storage account.

## Required permissions

To apply a virtual network rule to a storage account, the user must have the appropriate permissions for the subnets being added. Applying a rule can be performed by a [Storage Account Contributor](#) or a user that has been given permission to the `Microsoft.Network/virtualNetworks/subnets/joinViaServiceEndpoint/action` [Azure resource provider operation](#) via a custom Azure role.

Storage account and the virtual networks granted access may be in different subscriptions, including subscriptions that are a part of a different Azure AD tenant.

#### NOTE

Configuration of rules that grant access to subnets in virtual networks that are a part of a different Azure Active Directory tenant are currently only supported through PowerShell, CLI and REST APIs. Such rules cannot be configured through the Azure portal, though they may be viewed in the portal.

## Available virtual network regions

By default, service endpoints work between virtual networks and service instances in the same Azure region. When using service endpoints with Azure Storage, service endpoints also work between virtual networks and service instances in a [paired region](#). If you want to use a service endpoint to grant access to virtual networks in other regions, you must register the `AllowGlobalTagsForStorage` feature in the subscription of the virtual network. This capability is currently in public preview.

Service endpoints allow continuity during a regional failover and access to read-only geo-redundant storage (RA-GRS) instances. Network rules that grant access from a virtual network to a storage account also grant access to any RA-GRS instance.

When planning for disaster recovery during a regional outage, you should create the VNets in the paired region in advance. Enable service endpoints for Azure Storage, with network rules granting access from these alternative virtual networks. Then apply these rules to your geo-redundant storage accounts.

## Enabling access to virtual networks in other regions (preview)

#### IMPORTANT

This capability is currently in PREVIEW.

See the [Supplemental Terms of Use for Microsoft Azure Previews](#) for legal terms that apply to Azure features that are in beta, preview, or otherwise not yet released into general availability.

To enable access from a virtual network that is located in another region over service endpoints, register the `AllowGlobalTagsForStorage` feature in the subscription of the virtual network. All the subnets in the subscription that has the *AllowedGlobalTagsForStorage* feature enabled will no longer use a public IP address to communicate with any storage account. Instead, all the traffic from these subnets to storage accounts will use a private IP address as a source IP. As a result, any storage accounts that use IP network rules to permit traffic from those subnets will no longer have an effect.

#### NOTE

For updating the existing service endpoints to access a storage account in another region, perform an [update subnet](#) operation on the subnet after registering the subscription with the `AllowGlobalTagsForStorage` feature. Similarly, to go back to the old configuration, perform an [update subnet](#) operation after deregistering the subscription with the `AllowGlobalTagsForStorage` feature.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

During the preview you must use either PowerShell or the Azure CLI to enable this feature.

#### Managing virtual network rules

You can manage virtual network rules for storage accounts through the Azure portal, PowerShell, or CLIV2.

#### NOTE

If you registered the `AllowGlobalTagsForStorage` feature, and you want to enable access to your storage account from a virtual network/subnet in another Azure AD tenant, or in a region other than the region of the storage account or its paired region, then you must use PowerShell or the Azure CLI. The Azure portal does not show subnets in other Azure AD tenants or in regions other than the region of the storage account or its paired region, and hence cannot be used to configure access rules for virtual networks in other regions.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

1. Go to the storage account you want to secure.
2. Select on the settings menu called **Networking**.
3. Check that you've selected to allow access from **Selected networks**.
4. To grant access to a virtual network with a new network rule, under **Virtual networks**, select **Add existing virtual network**, select **Virtual networks** and **Subnets** options, and then select **Add**. To create a new virtual network and grant it access, select **Add new virtual network**. Provide the information necessary to create the new virtual network, and then select **Create**.

#### **NOTE**

If a service endpoint for Azure Storage wasn't previously configured for the selected virtual network and subnets, you can configure it as part of this operation.

Presently, only virtual networks belonging to the same Azure Active Directory tenant are shown for selection during rule creation. To grant access to a subnet in a virtual network belonging to another tenant, please use , PowerShell, CLI or REST APIs.

Even if you registered the `AllowGlobalTagsForStorageOnly` feature, subnets in regions other than the region of the storage account or its paired region aren't shown for selection. If you want to enable access to your storage account from a virtual network/subnet in a different region, use the instructions in the PowerShell or Azure CLI tabs.

5. To remove a virtual network or subnet rule, select ... to open the context menu for the virtual network or subnet, and select **Remove**.
6. select **Save** to apply your changes.

## Grant access from an internet IP range

You can use IP network rules to allow access from specific public internet IP address ranges by creating IP network rules. Each storage account supports up to 200 rules. These rules grant access to specific internet-based services and on-premises networks and blocks general internet traffic.

The following restrictions apply to IP address ranges.

- IP network rules are allowed only for **public internet** IP addresses.

IP address ranges reserved for private networks (as defined in [RFC 1918](#)) aren't allowed in IP rules.

Private networks include addresses that start with `10.**`, `172.16 - *172.31`, and `*192.168..`

- You must provide allowed internet address ranges using [CIDR notation](#) in the form `16.17.18.0/24` or as individual IP addresses like `16.17.18.19`.
- Small address ranges using "/31" or "/32" prefix sizes are not supported. These ranges should be configured using individual IP address rules.
- Only IPV4 addresses are supported for configuration of storage firewall rules.

IP network rules can't be used in the following cases:

- To restrict access to clients in same Azure region as the storage account.

IP network rules have no effect on requests originating from the same Azure region as the storage account. Use [Virtual network rules](#) to allow same-region requests.

- To restrict access to clients in a [paired region](#) which are in a VNet that has a service endpoint.
- To restrict access to Azure services deployed in the same region as the storage account.

Services deployed in the same region as the storage account use private Azure IP addresses for communication. Thus, you can't restrict access to specific Azure services based on their public outbound IP address range.

### Configuring access from on-premises networks

To grant access from your on-premises networks to your storage account with an IP network rule, you must identify the internet facing IP addresses used by your network. Contact your network administrator for help.

If you are using [ExpressRoute](#) from your premises, for public peering or Microsoft peering, you will need to identify the NAT IP addresses that are used. For public peering, each ExpressRoute circuit by default uses two NAT IP addresses applied to Azure service traffic when the traffic enters the Microsoft Azure network backbone. For Microsoft peering, the NAT IP addresses used are either customer provided or are provided by the service provider. To allow access to your service resources, you must allow these public IP addresses in the resource IP firewall setting. To find your public peering ExpressRoute circuit IP addresses, [open a support ticket with ExpressRoute](#) via the Azure portal. Learn more about [NAT for ExpressRoute public and Microsoft peering](#).

## Managing IP network rules

You can manage IP network rules for storage accounts through the Azure portal, PowerShell, or CLIV2.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

1. Go to the storage account you want to secure.
2. Select on the settings menu called **Networking**.
3. Check that you've selected to allow access from **Selected networks**.
4. To grant access to an internet IP range, enter the IP address or address range (in CIDR format) under **Firewall > Address Range**.
5. To remove an IP network rule, select the trash can icon next to the address range.
6. Select **Save** to apply your changes.

## Grant access from Azure resource instances

In some cases, an application might depend on Azure resources that cannot be isolated through a virtual network or an IP address rule. However, you'd still like to secure and restrict storage account access to only your application's Azure resources. You can configure storage accounts to allow access to specific resource instances of some Azure services by creating a resource instance rule.

The types of operations that a resource instance can perform on storage account data is determined by the Azure role assignments of the resource instance. Resource instances must be from the same tenant as your storage account, but they can belong to any subscription in the tenant.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

You can add or remove resource network rules in the Azure portal.

1. Sign in to the [Azure portal](#) to get started.
2. Locate your storage account and display the account overview.
3. Select **Networking** to display the configuration page for networking.
4. Under **Firewalls and virtual networks**, for **Selected networks**, select to allow access.
5. Scroll down to find **Resource instances**, and in the **Resource type** dropdown list, choose the resource type of your resource instance.
6. In the **Instance name** dropdown list, choose the resource instance. You can also choose to include all resource instances in the active tenant, subscription, or resource group.

7. Select **Save** to apply your changes. The resource instance appears in the **Resource instances** section of the network settings page.

To remove the resource instance, select the delete icon (  ) next to the resource instance.

## Grant access to trusted Azure services

Some Azure services operate from networks that can't be included in your network rules. You can grant a subset of such trusted Azure services access to the storage account, while maintaining network rules for other apps. These trusted services will then use strong authentication to securely connect to your storage account.

You can grant access to trusted Azure services by creating a network rule exception. For step-by-step guidance, see the [Manage exceptions](#) section of this article.

When you grant access to trusted Azure services, you grant the following types of access:

- Trusted access for select operations to resources that are registered in your subscription.
- Trusted access to resources based on a managed identity.

### Trusted access for resources registered in your subscription

Resources of some services, **when registered in your subscription**, can access your storage account **in the same subscription** for select operations, such as writing logs or backup. The following table describes each service and the operations allowed.

SERVICE	RESOURCE PROVIDER NAME	OPERATIONS ALLOWED
Azure Backup	Microsoft.RecoveryServices	Run backups and restores of unmanaged disks in IAAS virtual machines. (not required for managed disks). <a href="#">Learn more</a> .
Azure Data Box	Microsoft.DataBox	Enables import of data to Azure using Data Box. <a href="#">Learn more</a> .
Azure DevTest Labs	Microsoft.DevTestLab	Custom image creation and artifact installation. <a href="#">Learn more</a> .
Azure Event Grid	Microsoft.EventGrid	Enable Blob Storage event publishing and allow Event Grid to publish to storage queues. Learn about <a href="#">blob storage events</a> and <a href="#">publishing to queues</a> .
Azure Event Hubs	Microsoft.EventHub	Archive data with Event Hubs Capture. <a href="#">Learn More</a> .
Azure File Sync	Microsoft.StorageSync	Enables you to transform your on-prem file server to a cache for Azure File shares. Allowing for multi-site sync, fast disaster-recovery, and cloud-side backup. <a href="#">Learn more</a>
Azure HDInsight	Microsoft.HDInsight	Provision the initial contents of the default file system for a new HDInsight cluster. <a href="#">Learn more</a> .

Service	Resource provider name	Operations allowed
Azure Import Export	Microsoft.ImportExport	Enables import of data to Azure Storage or export of data from Azure Storage using the Azure Storage Import/Export service. <a href="#">Learn more</a> .
Azure Monitor	Microsoft.Insights	Allows writing of monitoring data to a secured storage account, including resource logs, Azure Active Directory sign-in and audit logs, and Microsoft Intune logs. <a href="#">Learn more</a> .
Azure Networking	Microsoft.Network	Store and analyze network traffic logs, including through the Network Watcher and Traffic Analytics services. <a href="#">Learn more</a> .
Azure Site Recovery	Microsoft.SiteRecovery	Enable replication for disaster-recovery of Azure IaaS virtual machines when using firewall-enabled cache, source, or target storage accounts. <a href="#">Learn more</a> .

### Trusted access based on a managed identity

The following table lists services that can have access to your storage account data if the resource instances of those services are given the appropriate permission.

If your account does not have the hierarchical namespace feature enabled on it, you can grant permission, by explicitly assigning an Azure role to the [managed identity](#) for each resource instance. In this case, the scope of access for the instance corresponds to the Azure role assigned to the managed identity.

You can use the same technique for an account that has the hierarchical namespace feature enable on it. However, you don't have to assign an Azure role if you add the managed identity to the access control list (ACL) of any directory or blob contained in the storage account. In that case, the scope of access for the instance corresponds to the directory or file to which the managed identity has been granted access. You can also combine Azure roles and ACLs together. To learn more about how to combine them together to grant access, see [Access control model in Azure Data Lake Storage Gen2](#).

#### TIP

The recommended way to grant access to specific resources is to use resource instance rules. To grant access to specific resource instances, see the [Grant access from Azure resource instances](#) section of this article.

Service	Resource provider name	Purpose
Azure API Management	Microsoft.ApiManagement/service	Enables Api Management service access to storage accounts behind firewall using policies. <a href="#">Learn more</a> .
Azure Cache for Redis	Microsoft.Cache/Redis	Allows access to storage accounts through Azure Cache for Redis. <a href="#">Learn more</a>

Service	Resource Provider Name	Purpose
Azure Cognitive Search	Microsoft.Search/searchServices	Enables Cognitive Search services to access storage accounts for indexing, processing and querying.
Azure Cognitive Services	Microsoft.CognitiveService/accounts	Enables Cognitive Services to access storage accounts. <a href="#">Learn more</a> .
Azure Container Registry Tasks	Microsoft.ContainerRegistry/registries	ACR Tasks can access storage accounts when building container images.
Azure Data Factory	Microsoft.DataFactory/factories	Allows access to storage accounts through the ADF runtime.
Azure Data Share	Microsoft.DataShare/accounts	Allows access to storage accounts through Data Share.
Azure DevTest Labs	Microsoft.DevTestLab/labs	Allows access to storage accounts through DevTest Labs.
Azure Event Grid	Microsoft.EventGrid/topics	Allows access to storage accounts through the Azure Event Grid.
Azure Healthcare APIs	Microsoft.HealthcareApis/services	Allows access to storage accounts through Azure Healthcare APIs.
Azure IoT Central Applications	Microsoft.IoTCentral/IoTApps	Allows access to storage accounts through Azure IoT Central Applications.
Azure IoT Hub	Microsoft.Devices/IotHubs	Allows data from an IoT hub to be written to Blob storage. <a href="#">Learn more</a>
Azure Logic Apps	Microsoft.Logic/workflows	Enables logic apps to access storage accounts. <a href="#">Learn more</a> .
Azure Machine Learning Service	Microsoft.MachineLearningServices	Authorized Azure Machine Learning workspaces write experiment output, models, and logs to Blob storage and read the data. <a href="#">Learn more</a> .
Azure Media Services	Microsoft.Media/mediaservices	Allows access to storage accounts through Media Services.
Azure Migrate	Microsoft.Migrate/migrateprojects	Allows access to storage accounts through Azure Migrate.
Microsoft Purview	Microsoft.Purview/accounts	Allows Microsoft Purview to access storage accounts.
Azure Remote Rendering	Microsoft.MixedReality/remoteRenderingAccounts	Allows access to storage accounts through Remote Rendering.
Azure Site Recovery	Microsoft.RecoveryServices/vaults	Allows access to storage accounts through Site Recovery.

Service	Resource provider name	Purpose
Azure SQL Database	Microsoft.Sql	Allows writing audit data to storage accounts behind firewall.
Azure Synapse Analytics	Microsoft.Sql	Allows import and export of data from specific SQL databases using the COPY statement or PolyBase (in dedicated pool), or the openrowset function and external tables in serverless pool. <a href="#">Learn more</a> .
Azure Stream Analytics	Microsoft.StreamAnalytics	Allows data from a streaming job to be written to Blob storage. <a href="#">Learn more</a> .
Azure Synapse Analytics	Microsoft.Synapse/workspaces	Enables access to data in Azure Storage from Azure Synapse Analytics.

## Grant access to storage analytics

In some cases, access to read resource logs and metrics is required from outside the network boundary. When configuring trusted services access to the storage account, you can allow read-access for the log files, metrics tables, or both by creating a network rule exception. For step-by-step guidance, see the **Manage exceptions** section below. To learn more about working with storage analytics, see [Use Azure Storage analytics to collect logs and metrics data](#).

## Manage exceptions

You can manage network rule exceptions through the Azure portal, PowerShell, or Azure CLI v2.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

1. Go to the storage account you want to secure.
2. Select on the settings menu called **Networking**.
3. Check that you've selected to allow access from **Selected networks**.
4. Under **Exceptions**, select the exceptions you wish to grant.
5. Select **Save** to apply your changes.

## Next steps

Learn more about Azure Network service endpoints in [Service endpoints](#).

Dig deeper into Azure Storage security in [Azure Storage security guide](#).

# Enforce a minimum required version of Transport Layer Security (TLS) for requests to a storage account

8/22/2022 • 16 minutes to read • [Edit Online](#)

Communication between a client application and an Azure Storage account is encrypted using Transport Layer Security (TLS). TLS is a standard cryptographic protocol that ensures privacy and data integrity between clients and services over the Internet. For more information about TLS, see [Transport Layer Security](#).

Azure Storage currently supports three versions of the TLS protocol: 1.0, 1.1, and 1.2. Azure Storage uses TLS 1.2 on public HTTPS endpoints, but TLS 1.0 and TLS 1.1 are still supported for backward compatibility.

Azure Storage accounts permit clients to send and receive data with the oldest version of TLS, TLS 1.0, and above. To enforce stricter security measures, you can configure your storage account to require that clients send and receive data with a newer version of TLS. If a storage account requires a minimum version of TLS, then any requests made with an older version will fail.

This article describes how to use a DRAG (Detection-Remediation-Audit-Governance) framework to continuously manage secure TLS for your storage accounts.

For information about how to specify a particular version of TLS when sending a request from a client application, see [Configure Transport Layer Security \(TLS\) for a client application](#).

## Detect the TLS version used by client applications

When you enforce a minimum TLS version for your storage account, you risk rejecting requests from clients that are sending data with an older version of TLS. To understand how configuring the minimum TLS version may affect client applications, Microsoft recommends that you enable logging for your Azure Storage account and analyze the logs after an interval of time to detect what versions of TLS client applications are using.

To log requests to your Azure Storage account and determine the TLS version used by the client, you can use Azure Storage logging in Azure Monitor (preview). For more information, see [Monitor Azure Storage](#).

Azure Storage logging in Azure Monitor supports using log queries to analyze log data. To query logs, you can use an Azure Log Analytics workspace. To learn more about log queries, see [Tutorial: Get started with Log Analytics queries](#).

To log Azure Storage data with Azure Monitor and analyze it with Azure Log Analytics, you must first create a diagnostic setting that indicates what types of requests and for which storage services you want to log data. Azure Storage logs in Azure Monitor is in public preview and is available for preview testing in all public cloud regions. This preview enables logs for blobs (including Azure Data Lake Storage Gen2), files, queues, and tables. To create a diagnostic setting in the Azure portal, follow these steps:

1. Create a new Log Analytics workspace in the subscription that contains your Azure Storage account. After you configure logging for your storage account, the logs will be available in the Log Analytics workspace. For more information, see [Create a Log Analytics workspace in the Azure portal](#).
2. Navigate to your storage account in the Azure portal.
3. In the Monitoring section, select **Diagnostic settings (preview)**.
4. Select the Azure Storage service for which you want to log requests. For example, choose **Blob** to log

requests to Blob storage.

5. Select **Add diagnostic setting**.
6. Provide a name for the diagnostic setting.
7. Under **Category details**, in the **log** section, choose which types of requests to log. You can log read, write, and delete requests. For example, choosing **StorageRead** and **StorageWrite** will log read and write requests to the selected service.
8. Under **Destination details**, select **Send to Log Analytics**. Select your subscription and the Log Analytics workspace you created earlier, as shown in the following image.

The screenshot shows the 'Diagnostics settings' blade in the Azure portal. At the top, there are buttons for Save, Discard, Delete, and Provide feedback. Below that, a descriptive text explains what a diagnostic setting is. The main area has two sections: 'Category details' and 'Destination details'. In 'Category details', under 'log', 'StorageRead' and 'StorageWrite' are checked, while 'StorageDelete' is unchecked. Under 'metric', 'Transaction' is unchecked. In 'Destination details', the 'Send to Log Analytics' checkbox is checked, and the 'Subscription' dropdown is set to 'Azure Storage content development and testing'. The 'Log Analytics workspace' dropdown is set to 'TlsLogAnalytics ( eastus )'. There are also two unchecked checkboxes for 'Archive to a storage account' and 'Stream to an event hub'.

After you create the diagnostic setting, requests to the storage account are subsequently logged according to that setting. For more information, see [Create diagnostic setting to collect resource logs and metrics in Azure](#).

For a reference of fields available in Azure Storage logs in Azure Monitor, see [Resource logs \(preview\)](#).

### Query logged requests by TLS version

Azure Storage logs in Azure Monitor include the TLS version used to send a request to a storage account. Use the **TlsVersion** property to check the TLS version of a logged request.

To determine how many requests were made against Blob storage with different versions of TLS over the past seven days, open your Log Analytics workspace. Next, paste the following query into a new log query and run it. Remember to replace the placeholder values in brackets with your own values:

```
StorageBlobLogs
| where TimeGenerated > ago(7d) and AccountName == "<account-name>"
| summarize count() by TlsVersion
```

The results show the count of the number of requests made with each version of TLS:

Results		Chart	Columns
<b>Completed</b>			
TlsVersion	▼	count_	▼
TLS 1.2		146	
TlsVersion	TLS 1.2		
count_	146		
TLS 1.1		16	
TlsVersion	TLS 1.1		
count_	16		

## Query logged requests by caller IP address and user agent header

Azure Storage logs in Azure Monitor also include the caller IP address and user agent header to help you to evaluate which client applications accessed the storage account. You can analyze these values to decide whether client applications must be updated to use a newer version of TLS, or whether it's acceptable to fail a client's request if it is not sent with the minimum TLS version.

To determine which clients made requests with a version of TLS older than TLS 1.2 over the past seven days, paste the following query into a new log query and run it. Remember to replace the placeholder values in brackets with your own values:

```
StorageBlobLogs
| where TimeGenerated > ago(7d) and AccountName == "<account-name>" and TlsVersion != "TLS 1.2"
| project TlsVersion, CallerIpAddress, UserAgentHeader
```

## Remediate security risks with a minimum version of TLS

When you are confident that traffic from clients using older versions of TLS is minimal, or that it's acceptable to fail requests made with an older version of TLS, then you can begin enforcement of a minimum TLS version on your storage account. Requiring that clients use a minimum version of TLS to make requests against a storage account is part of a strategy to minimize security risks to your data.

### IMPORTANT

If you are using a service that connects to Azure Storage, make sure that that service is using the appropriate version of TLS to send requests to Azure Storage before you set the required minimum version for a storage account.

## Configure the minimum TLS version for a storage account

To configure the minimum TLS version for a storage account, set the **MinimumTlsVersion** version for the account. This property is available for all storage accounts that are created with the Azure Resource Manager deployment model. For more information about the Azure Resource Manager deployment model, see [Storage account overview](#).

The default value of the **MinimumTlsVersion** property is different depending on how you set it. When you create a storage account with the Azure portal, the minimum TLS version is set to 1.2 by default. When you create a storage account with PowerShell, Azure CLI, or an Azure Resource Manager template, the **MinimumTlsVersion** property is not set by default and does not return a value until you explicitly set it.

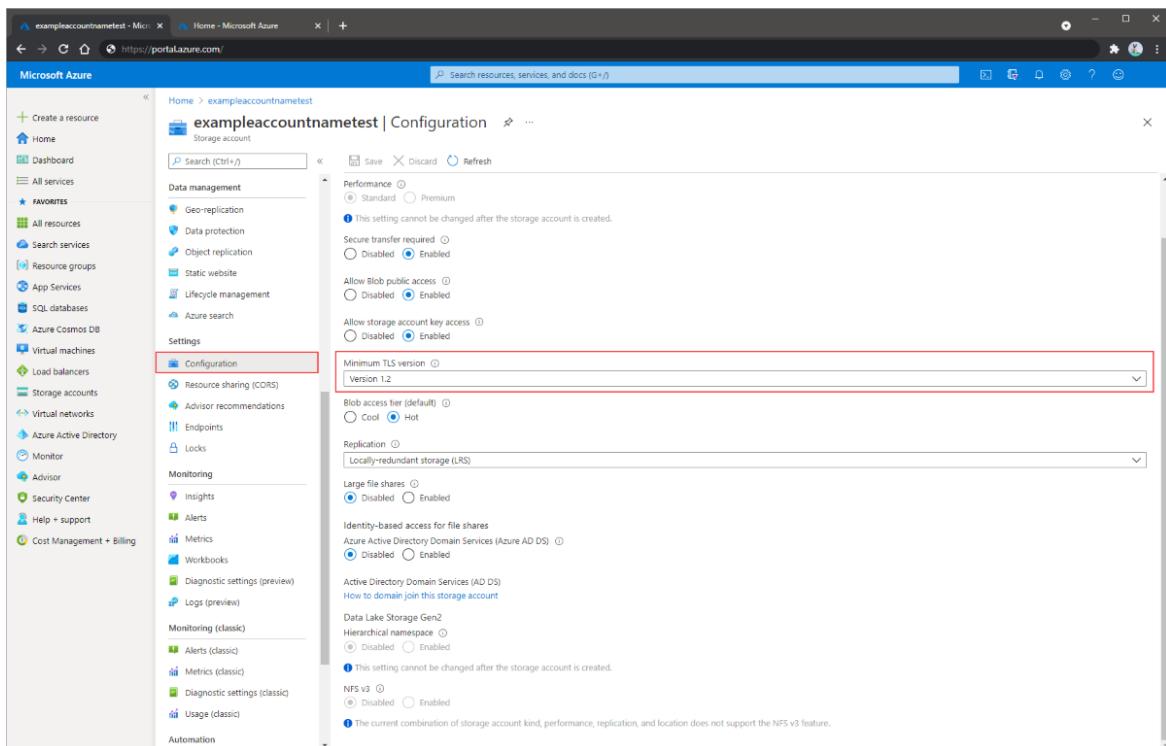
When the **MinimumTlsVersion** property is not set, its value may be displayed as either **null** or an empty string, depending on the context. The storage account will permit requests sent with TLS version 1.0 or greater if the property is not set.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Template](#)

When you create a storage account with the Azure portal, the minimum TLS version is set to 1.2 by default.

To configure the minimum TLS version for an existing storage account with the Azure portal, follow these steps:

1. Navigate to your storage account in the Azure portal.
2. Under **Settings**, select **Configuration**.
3. Under **Minimum TLS version**, use the drop-down to select the minimum version of TLS required to access data in this storage account.



#### NOTE

After you update the minimum TLS version for the storage account, it may take up to 30 seconds before the change is fully propagated.

Configuring the minimum TLS version requires version 2019-04-01 or later of the Azure Storage resource provider. For more information, see [Azure Storage Resource Provider REST API](#).

#### Check the minimum required TLS version for multiple accounts

To check the minimum required TLS version across a set of storage accounts with optimal performance, you can use the Azure Resource Graph Explorer in the Azure portal. To learn more about using the Resource Graph Explorer, see [Quickstart: Run your first Resource Graph query using Azure Resource Graph Explorer](#).

Running the following query in the Resource Graph Explorer returns a list of storage accounts and displays the minimum TLS version for each account:

```
resources
| where type == 'Microsoft.Storage/storageAccounts'
| extend minimumTlsVersion = parse_json(properties).minimumTlsVersion
| project subscriptionId, resourceGroup, name, minimumTlsVersion
```

## Test the minimum TLS version from a client

To test that the minimum required TLS version for a storage account forbids calls made with an older version, you can configure a client to use an older version of TLS. For more information about configuring a client to use a specific version of TLS, see [Configure Transport Layer Security \(TLS\) for a client application](#).

When a client accesses a storage account using a TLS version that does not meet the minimum TLS version configured for the account, Azure Storage returns error code 400 error (Bad Request) and a message indicating that the TLS version that was used is not permitted for making requests against this storage account.

### NOTE

When you configure a minimum TLS version for a storage account, that minimum version is enforced at the application layer. Tools that attempt to determine TLS support at the protocol layer may return TLS versions in addition to the minimum required version when run directly against the storage account endpoint.

## Use Azure Policy to audit for compliance

If you have a large number of storage accounts, you may want to perform an audit to make sure that all accounts are configured for the minimum version of TLS that your organization requires. To audit a set of storage accounts for their compliance, use Azure Policy. Azure Policy is a service that you can use to create, assign, and manage policies that apply rules to Azure resources. Azure Policy helps you to keep those resources compliant with your corporate standards and service level agreements. For more information, see [Overview of Azure Policy](#).

### Create a policy with an Audit effect

Azure Policy supports effects that determine what happens when a policy rule is evaluated against a resource. The Audit effect creates a warning when a resource is not in compliance, but does not stop the request. For more information about effects, see [Understand Azure Policy effects](#).

To create a policy with an Audit effect for the minimum TLS version with the Azure portal, follow these steps:

1. In the Azure portal, navigate to the Azure Policy service.
2. Under the **Authoring** section, select **Definitions**.
3. Select **Add policy definition** to create a new policy definition.
4. For the **Definition location** field, select the **More** button to specify where the audit policy resource is located.
5. Specify a name for the policy. You can optionally specify a description and category.
6. Under **Policy rule**, add the following policy definition to the **policyRule** section.

```
{
 "policyRule": {
 "if": {
 "allOf": [
 {
 "field": "type",
 "equals": "Microsoft.Storage/storageAccounts"
 },
 {
 "anyOf": [
 {
 "field": "Microsoft.Storage/storageAccounts/minimumTlsVersion",
 "notEquals": "TLS1_2"
 },
 {
 "field": "Microsoft.Storage/storageAccounts/minimumTlsVersion",
 "exists": "false"
 }
]
 }
],
 "then": {
 "effect": "audit"
 }
 }
 }
}
```

## 7. Save the policy.

### Assign the policy

Next, assign the policy to a resource. The scope of the policy corresponds to that resource and any resources beneath it. For more information on policy assignment, see [Azure Policy assignment structure](#).

To assign the policy with the Azure portal, follow these steps:

1. In the Azure portal, navigate to the Azure Policy service.
2. Under the **Authoring** section, select **Assignments**.
3. Select **Assign policy** to create a new policy assignment.
4. For the **Scope** field, select the scope of the policy assignment.
5. For the **Policy definition** field, select the **More** button, then select the policy you defined in the previous section from the list.
6. Provide a name for the policy assignment. The description is optional.
7. Leave **Policy enforcement** set to *Enabled*. This setting has no effect on the audit policy.
8. Select **Review + create** to create the assignment.

### View compliance report

After you've assigned the policy, you can view the compliance report. The compliance report for an audit policy provides information on which storage accounts are not in compliance with the policy. For more information, see [Get policy compliance data](#).

It may take several minutes for the compliance report to become available after the policy assignment is created.

To view the compliance report in the Azure portal, follow these steps:

1. In the Azure portal, navigate to the Azure Policy service.
2. Select **Compliance**.

3. Filter the results for the name of the policy assignment that you created in the previous step. The report shows how many resources are not in compliance with the policy.
4. You can drill down into the report for additional details, including a list of storage accounts that are not in compliance.

The screenshot shows the 'Assignment Details' page for a policy named 'TLS policy 1 Assignment'. The page displays the following information:

- Name:** TLS policy 1 Assignment
- Scope:** Azure Storage content development and testing/storagesamples-rg
- Description:** --
- Excluded scopes:** 0
- Assignment ID:** <resource-id>
- Definition:** TLS policy 1

Under 'Selected Scopes', it shows 6 selected subscriptions. The 'Compliance state' section indicates 'Non-compliant' with a red 'X' icon and 4% overall resource compliance (2 out of 55). The 'Non-compliant resources' section shows 53 out of 55 resources. The 'Events (last 7 days)' section shows 0 Audit, 0 Append, 0 Modify, 0 Deny, and 0 Deploy events.

The 'Resource compliance' tab is selected, displaying a table of non-compliant resources. The table has columns: Name, Compliance state, Compliance reason, Resource Type, Location, Scope, and a 'Details' column. The table lists four resources:

Name	Compliance state	Compliance reason	Resource Type	Location	Scope
storagesamplespolicy2	Non-compliant	Details	Microsoft.Storage/st...	East US	Azure Storage content d...
storagesamplestatic	Non-compliant	Details	Microsoft.Storage/st...	West US 2	Azure Storage content d...
storagesamplessecurity	Non-compliant	Details	Microsoft.Storage/st...	West US 2	Azure Storage content d...
storagesamplesrgdiag...	Non-compliant	Details	Microsoft.Storage/st...	West US 2	Azure Storage content d...

## Use Azure Policy to enforce the minimum TLS version

Azure Policy supports cloud governance by ensuring that Azure resources adhere to requirements and standards. To enforce a minimum TLS version requirement for the storage accounts in your organization, you can create a policy that prevents the creation of a new storage account that sets the minimum TLS requirement to an older version of TLS than that which is dictated by the policy. This policy will also prevent all configuration changes to an existing account if the minimum TLS version setting for that account is not compliant with the policy.

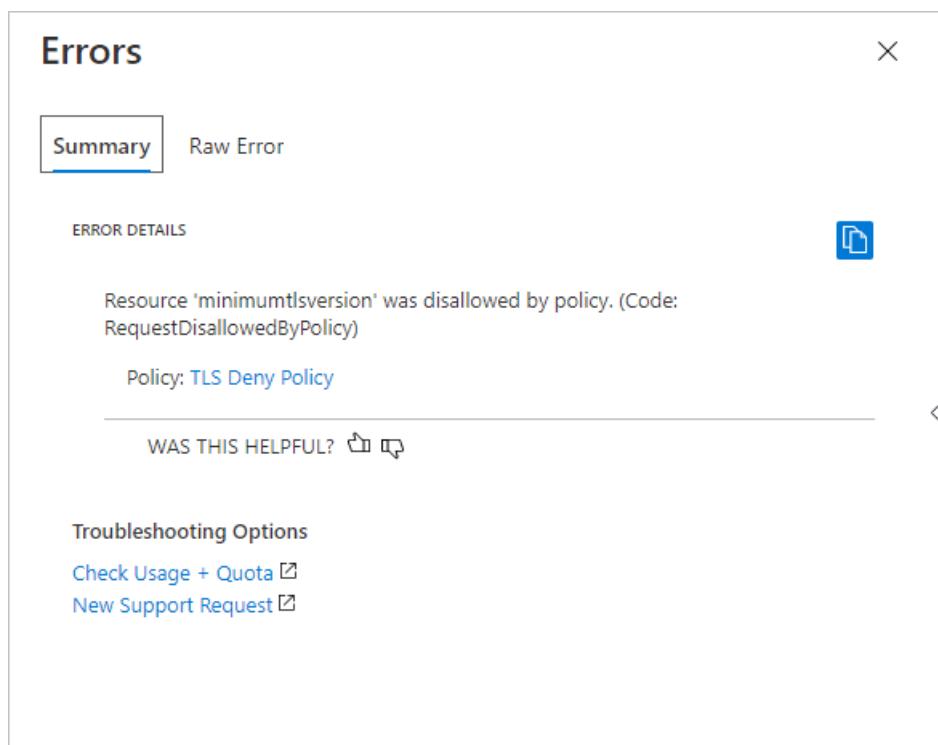
The enforcement policy uses the Deny effect to prevent a request that would create or modify a storage account so that the minimum TLS version no longer adheres to your organization's standards. For more information about effects, see [Understand Azure Policy effects](#).

To create a policy with a Deny effect for a minimum TLS version that is less than TLS 1.2, follow the same steps described in [Use Azure Policy to audit for compliance](#), but provide the following JSON in the **policyRule** section of the policy definition:

```
{
 "policyRule": {
 "if": {
 "allOf": [
 {
 "field": "type",
 "equals": "Microsoft.Storage/storageAccounts"
 },
 {
 "anyOf": [
 {
 "field": "Microsoft.Storage/storageAccounts/minimumTlsVersion",
 "notEquals": "TLS1_2"
 },
 {
 "field": "Microsoft.Storage/storageAccounts/minimumTlsVersion",
 "exists": "false"
 }
]
 }
],
 "then": {
 "effect": "deny"
 }
 }
 }
}
```

After you create the policy with the Deny effect and assign it to a scope, a user cannot create a storage account with a minimum TLS version that is older than 1.2. Nor can a user make any configuration changes to an existing storage account that currently requires a minimum TLS version that is older than 1.2. Attempting to do so results in an error. The required minimum TLS version for the storage account must be set to 1.2 to proceed with account creation or configuration.

The following image shows the error that occurs if you try to create a storage account with the minimum TLS version set to TLS 1.0 (the default for a new account) when a policy with a Deny effect requires that the minimum TLS version be set to TLS 1.2.



## Permissions necessary to require a minimum version of TLS

To set the **MinimumTlsVersion** property for the storage account, a user must have permissions to create and manage storage accounts. Azure role-based access control (Azure RBAC) roles that provide these permissions include the **Microsoft.Storage/storageAccounts/write** or **Microsoft.Storage/storageAccounts/\*** action. Built-in roles with this action include:

- The Azure Resource Manager [Owner](#) role
- The Azure Resource Manager [Contributor](#) role
- The [Storage Account Contributor](#) role

These roles do not provide access to data in a storage account via Azure Active Directory (Azure AD). However, they include the **Microsoft.Storage/storageAccounts/listkeys/action**, which grants access to the account access keys. With this permission, a user can use the account access keys to access all data in a storage account.

Role assignments must be scoped to the level of the storage account or higher to permit a user to require a minimum version of TLS for the storage account. For more information about role scope, see [Understand scope for Azure RBAC](#).

Be careful to restrict assignment of these roles only to those who require the ability to create a storage account or update its properties. Use the principle of least privilege to ensure that users have the fewest permissions that they need to accomplish their tasks. For more information about managing access with Azure RBAC, see [Best practices for Azure RBAC](#).

### NOTE

The classic subscription administrator roles Service Administrator and Co-Administrator include the equivalent of the Azure Resource Manager [Owner](#) role. The [Owner](#) role includes all actions, so a user with one of these administrative roles can also create and manage storage accounts. For more information, see [Classic subscription administrator roles, Azure roles, and Azure AD administrator roles](#).

## Network considerations

When a client sends a request to storage account, the client establishes a connection with the public endpoint of the storage account first, before processing any requests. The minimum TLS version setting is checked after the connection is established. If the request uses an earlier version of TLS than that specified by the setting, the connection will continue to succeed, but the request will eventually fail. For more information about public endpoints for Azure Storage, see [Resource URI syntax](#).

## Next steps

- [Configure Transport Layer Security \(TLS\) for a client application](#)
- [Security recommendations for Blob storage](#)

# Configure Transport Layer Security (TLS) for a client application

8/22/2022 • 2 minutes to read • [Edit Online](#)

For security purposes, an Azure Storage account may require that clients use a minimum version of Transport Layer Security (TLS) to send requests. Calls to Azure Storage will fail if the client is using a version of TLS that is lower than the minimum required version. For example, if a storage account requires TLS 1.2, then a request sent by a client who is using TLS 1.1 will fail.

This article describes how to configure a client application to use a particular version of TLS. For information about how to configure a minimum required version of TLS for an Azure Storage account, see [Configure minimum required version of Transport Layer Security \(TLS\) for a storage account](#).

## Configure the client TLS version

In order for a client to send a request with a particular version of TLS, the operating system must support that version.

The following examples show how to set the client's TLS version to 1.2 from PowerShell or .NET. The .NET Framework used by the client must support TLS 1.2. For more information, see [Support for TLS 1.2](#).

- [PowerShell](#)
- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

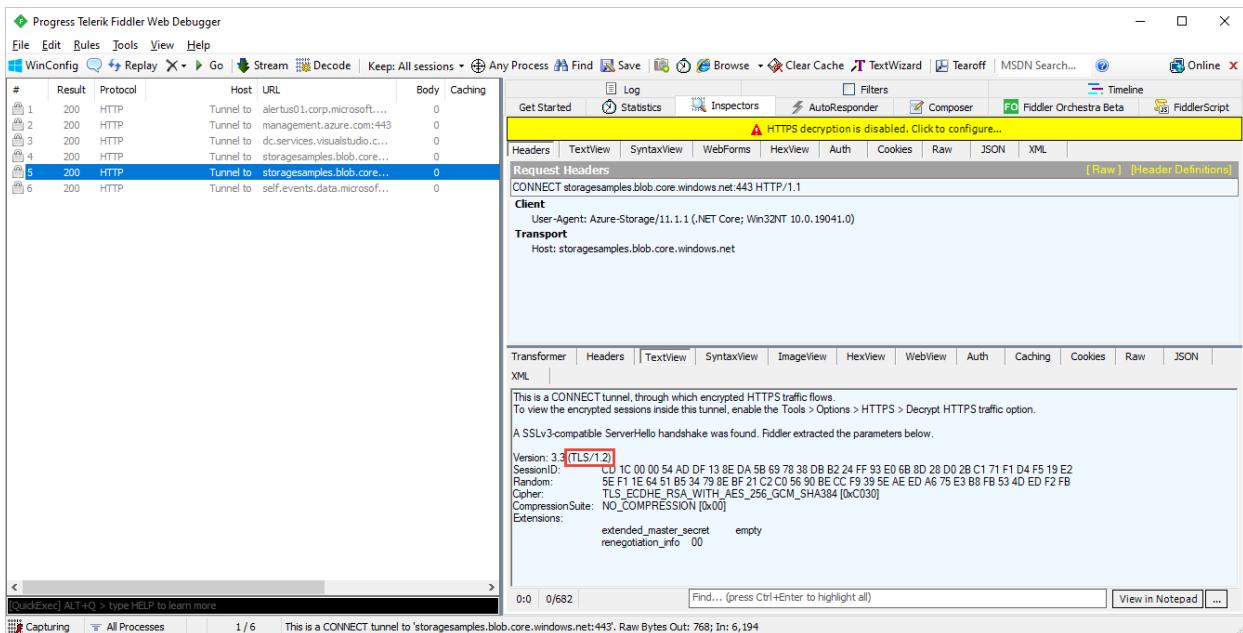
The following sample shows how to enable TLS 1.2 in a PowerShell client:

```
Set the TLS version used by the PowerShell client to TLS 1.2.
[System.Net.ServicePointManager]::SecurityProtocol = [System.Net.SecurityProtocolType]::Tls12;

Create a new container.
$storageAccount = Get-AzStorageAccount -ResourceGroupName $rgName -Name $accountName
$cxt = $storageAccount.Context
New-AzStorageContainer -Name "sample-container" -Context $cxt
```

## Verify the TLS version used by a client

To verify that the specified version of TLS was used by the client to send a request, you can use [Fiddler](#) or a similar tool. Open Fiddler to start capturing client network traffic, then execute one of the examples in the previous section. Look at the Fiddler trace to confirm that the correct version of TLS was used to send the request, as shown in the following image.



## Next steps

- Configure minimum required version of Transport Layer Security (TLS) for a storage account
- Security recommendations for Blob storage

# Configure network routing preference for Azure Storage

8/22/2022 • 4 minutes to read • [Edit Online](#)

This article describes how you can configure the network routing preference and route-specific endpoints for your storage account.

The network routing preference specifies how network traffic is routed to your account from clients over the internet. Route-specific endpoints are new endpoints that Azure Storage creates for your storage account. These endpoints route traffic over a desired path without changing your default routing preference. To learn more, see [Network routing preference for Azure Storage](#).

## Configure the routing preference for the default public endpoint

By default, the routing preference for the public endpoint of the storage account is set to Microsoft global network. You can choose between the Microsoft global network and Internet routing as the default routing preference for the public endpoint of your storage account. To learn more about the difference between these two types of routing, see [Network routing preference for Azure Storage](#).

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To change your routing preference to Internet routing:

1. Sign in to the [Azure portal](#).
2. Navigate to your storage account in the portal.
3. Under **Security + networking**, choose **Networking**.
4. In the **Firewalls and virtual networks** tab, under **Network Routing**, change the **Routing preference** setting to **Internet routing**.
5. Click **Save**.

The screenshot shows the 'Networking' blade in the Azure Storage settings. The 'Firewalls and virtual networks' tab is selected. Under 'Network Routing', the 'Routing preference' dropdown is set to 'Internet routing', which is highlighted with a red box. The 'Allow access from' section has 'Selected networks' selected. There is a note about configuring network security and a link to learn more. At the bottom, there are options to publish route-specific endpoints for Microsoft network routing or Internet routing.

Firewalls and virtual networks    Private endpoint connections

Save   Discard   Refresh

Allow access from

All networks  Selected networks

Configure network security for your storage accounts. [Learn more](#)

Network Routing

Determine how you would like to route your traffic as it travels from its source to an Azure endpoint. Microsoft routing is recommended for most customers.

Routing preference \* [\(i\)](#)

Microsoft network routing  Internet routing

Publish route-specific endpoints [\(i\)](#)

Microsoft network routing  
 Internet routing

## Configure a route-specific endpoint

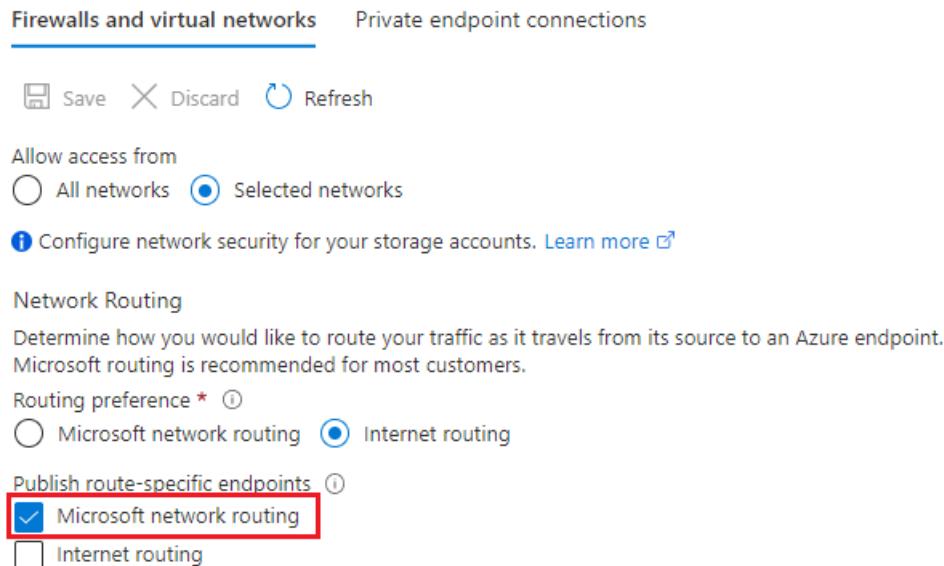
You can also configure a route-specific endpoint. For example, you can set the routing preference for the default endpoint to *Internet routing*, and then publish a route-specific endpoint that enables traffic between clients on the internet and your storage account to be routed via the Microsoft global network.

This preference affects only the route-specific endpoint. This preference doesn't affect your default routing preference.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

1. Navigate to your storage account in the portal.
2. Under **Security + networking**, choose **Networking**.
3. In the **Firewalls and virtual networks** tab, under **Publish route-specific endpoints**, choose the routing preference of your route-specific endpoint, and then click **Save**.

The following image shows the **Microsoft network routing** option selected.



## Find the endpoint name for a route-specific endpoint

If you configured a route-specific endpoint, you can find the endpoint in the properties of your storage account.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

1. Under **Settings**, choose **Endpoints**.

2. The **Microsoft network routing** endpoint is shown for each service that supports routing preferences.

This image shows the endpoint for the blob and file services.

<b>Blob service</b>	
Resource ID	/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx/resourceGroups/contoso-resource-gr... 
Blob service	https://contoso.blob.core.windows.net/ 
Microsoft network routing	https://contoso-microsoftrouting.blob.core.windows.net/ 
<b>File service</b>	
Resource ID	/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx/resourceGroups/contoso-resource-gr... 
File service	https://contoso.file.core.windows.net/ 
Microsoft network routing	https://contoso-microsoftrouting.file.core.windows.net/ 

## See also

- [Network routing preference](#)
- [Configure Azure Storage firewalls and virtual networks](#)
- [Security recommendations for Blob storage](#)

# Configure Microsoft Defender for Storage

8/22/2022 • 4 minutes to read • [Edit Online](#)

Microsoft Defender for Storage provides an additional layer of security intelligence that detects unusual and potentially harmful attempts to access or exploit storage accounts. This layer of protection allows you to address threats without being a security expert or managing security monitoring systems.

Security alerts are triggered when anomalies in activity occur. These security alerts are integrated with [Microsoft Defender for Cloud](#), and are also sent via email to subscription administrators, with details of suspicious activity and recommendations on how to investigate and remediate threats.

The service ingests resource logs of read, write, and delete requests to Blob storage and to Azure Files for threat detection. To investigate alerts from Microsoft Defender for Cloud, you can view related storage activity using Storage Analytics Logging. For more information, see [Configure logging in Monitor a storage account in the Azure portal](#).

## Availability

Microsoft Defender for Storage is currently available for Blob storage, Azure Files, and Azure Data Lake Storage Gen2. Account types that support Microsoft Defender for Storage include general-purpose v2, block blob, and Blob storage accounts. Microsoft Defender for Storage is available in all public clouds and US government clouds, but not in other sovereign or Azure Government cloud regions.

Accounts with hierarchical namespaces enabled for Data Lake Storage support transactions using both the Azure Blob storage APIs and the Data Lake Storage APIs. Azure file shares support transactions over SMB.

For pricing details, including a free 30 day trial, see the [Microsoft Defender for Cloud pricing page](#).

The following list summarizes the availability of Microsoft Defender for Storage:

- Release state:
  - [Blob Storage](#) (general availability)
  - [Azure Files](#) (general availability)
  - Azure Data Lake Storage Gen2 (general availability)
- Clouds:  Commercial clouds
  - Azure Government
  - Azure China 21Vianet

## Set up Microsoft Defender for Cloud

You can configure Microsoft Defender for Storage in any of several ways, described in the following sections.

- [Microsoft Defender for Cloud](#)
- [Portal](#)
- [Template](#)
- [Azure Policy](#)
- [PowerShell](#)
- [Azure CLI](#)

Microsoft Defender for Storage is built into Microsoft Defender for Cloud. When you enable Microsoft Defender for Cloud's enhanced security features on your subscription, Microsoft Defender for Storage is automatically

enabled for all of your storage accounts. To enable or disable Defender for Storage for individual storage accounts under a specific subscription:

1. Launch **Microsoft Defender for Cloud** in the [Azure portal](#).
2. From Defender for Cloud's main menu, select **Environment settings**.
3. Select the subscription for which you want to enable or disable Microsoft Defender for Cloud.
4. Select **Enable all Microsoft Defender plans** to enable Microsoft Defender for Cloud in the subscription.
5. Under **Select Microsoft Defender plans by resource type**, locate the **Storage** row, and select **Enabled** in the **Plan** column.
6. Save your changes.

The screenshot shows the 'Settings | Defender plans' page in the Azure portal. On the left, there's a navigation sidebar with 'Defender plans' selected. The main area displays a table of resources and their configuration status. A yellow warning message at the top states: 'Defender for Cloud plans will be enabled on 155 resources in this subscription'. The table has columns for 'Microsoft Defender for', 'Resource quantity', 'Plan / Pricing', 'Configuration', and 'Status'. The 'Storage' row is highlighted with a red box. It shows '73 storage accounts' and '\$0.02/10k transactions'. The 'Status' column for 'Storage' has an 'On' button, which is highlighted with a blue box. Other rows include 'Security posture management' (Free plan, Off), 'Servers' (Plan 2 (\$15/Server/Month), On), 'App Service' (\$15/instance/Month, On), 'Databases' (Selected: 4/4, Fully configured, On), 'Containers' (2 container registries; 34 kubernetes core, \$7/VM core/Month, Off), 'Key Vault' (\$0.02/10k transactions, On), 'Resource Manager' (\$4/1M resource management operations, On), and 'DNS' (\$0.7/1M DNS queries, On).

Microsoft Defender for Storage is now enabled for all storage accounts in this subscription.

## Explore security anomalies

When storage activity anomalies occur, you receive an email notification with information about the suspicious security event. Details of the event include:

- The nature of the anomaly
- The storage account name
- The event time
- The storage type
- The potential causes
- The investigation steps
- The remediation steps

The email also includes details on possible causes and recommended actions to investigate and mitigate the potential threat.



## MEDIUM SEVERITY

Someone has accessed your Storage account 'mystorageaccount' from an unusual location.

### Activity details

Subscription ID	38647a8d69a5496eb07f2a124eb7
Storage account	mystorageaccount
Storage type	Blob
Container	mycontainer
Application	myTestApplication
IP address	13.60.60.99
Location	Washington, United States
Data center	scus
Date	May 17, 2018 7:50 UTC
Potential causes	Unauthorized access that exploits an opening in the firewall. Legitimate access from a new location
Investigation steps	<a href="#">For a full investigation, configure diagnostics logs for read, write, and delete</a>
Remediation steps	Be sure to follow the principle of "least privilege" and <a href="#">limit access to your data</a>

You can review and manage your current security alerts from Microsoft Defender for Cloud's [Security alerts tile](#). Select an alert for details and actions for investigating the current threat and addressing future threats.

Home > Security alert

**PREVIEW - Potential malware uploaded to a storage file share**

High Severity	Active Status	06/29/20, 03:06 (1 day ago) Activity time
Alert description		
Someone has uploaded potential malware to your Azure Storage account 'hasandemo3'.		
Affected resource		
hasandemo3 Storage account		
Remote Data Protection-Playground Subscription		
Intent		
• Lateral Movement		
Alert details		
Azure AD user	Authentication type	File share
N/A (Azure AD authentication was not used)	Shared access signature (SAS)	atp-share-38647a8d69a5496eb07f2a124eb7
User agent	Investigation steps	File
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/67.0.3396.99 Safari/537.36	<a href="#">View related storage activity using Storage Analytics ...</a>	atp-eicar-38647a8d69a5496eb07f2a124eb7
Client IP address	Operations types	Malware description
<client-ip-address>	PutRange	File was identified as malicious, MalwareFamily
Client location	Service type	Detection source
<client-location>	Azure Files	Team Cymru
Related entities		
✓  Azure resource (1)		
✓  IP (1) Includes Geo & Threat Intelligence		
✓  Network connection (1)		
<a href="#">Next: Take Action &gt;&gt;</a>		

## Security alerts

Alerts are generated by unusual and potentially harmful attempts to access or exploit storage accounts. For a list

of alerts for Azure Storage, see [Alerts for Azure Storage](#).

## Next steps

- [Introduction to Microsoft Defender for Storage](#)
- [Microsoft Defender for Cloud](#)
- [Logs in Azure Storage accounts](#)

# Apply an Azure Resource Manager lock to a storage account

8/22/2022 • 3 minutes to read • [Edit Online](#)

Microsoft recommends locking all of your storage accounts with an Azure Resource Manager lock to prevent accidental or malicious deletion of the storage account. There are two types of Azure Resource Manager resource locks:

- A **CannotDelete** lock prevents users from deleting a storage account, but permits reading and modifying its configuration.
- A **ReadOnly** lock prevents users from deleting a storage account or modifying its configuration, but permits reading the configuration.

For more information about Azure Resource Manager locks, see [Lock resources to prevent changes](#).

**Caution**

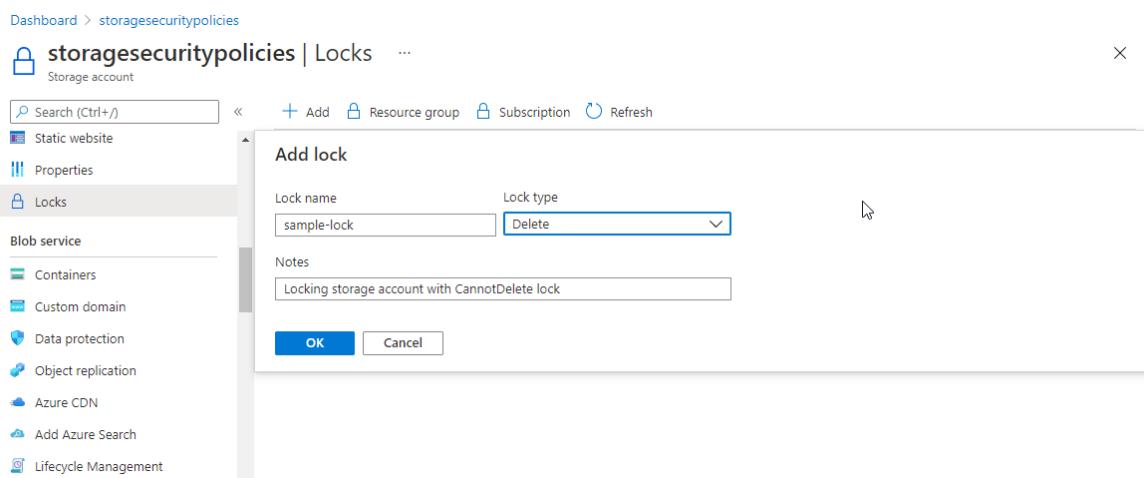
Locking a storage account does not protect containers or blobs within that account from being deleted or overwritten. For more information about how to protect blob data, see [Data protection overview](#).

## Configure an Azure Resource Manager lock

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To configure a lock on a storage account with the Azure portal, follow these steps:

1. Navigate to your storage account in the Azure portal.
2. Under the **Settings** section, select **Locks**.
3. Select **Add**.
4. Provide a name for the resource lock, and specify the type of lock. Add a note about the lock if desired.



## Authorizing data operations when a **ReadOnly** lock is in effect

When a **ReadOnly** lock is applied to a storage account, the [List Keys](#) operation is blocked for that storage

account. The **List Keys** operation is an HTTPS POST operation, and all POST operations are prevented when a **ReadOnly** lock is configured for the account. The **List Keys** operation returns the account access keys, which can then be used to read and write to any data in the storage account.

If a client is in possession of the account access keys at the time that the lock is applied to the storage account, then that client can continue to use the keys to access data. However, clients who do not have access to the keys will need to use Azure Active Directory (Azure AD) credentials to access blob or queue data in the storage account.

Users of the Azure portal may be affected when a **ReadOnly** lock is applied, if they have previously accessed blob or queue data in the portal with the account access keys. After the lock is applied, portal users will need to use Azure AD credentials to access blob or queue data in the portal. To do so, a user must have at least two RBAC roles assigned to them: the Azure Resource Manager Reader role at a minimum, and one of the Azure Storage data access roles. For more information, see one of the following articles:

- [Choose how to authorize access to blob data in the Azure portal](#)
- [Choose how to authorize access to queue data in the Azure portal](#)

Data in Azure Files or the Table service may become unaccessible to clients who have previously been accessing it with the account keys. As a best practice, if you must apply a **ReadOnly** lock to a storage account, then move your Azure Files and Table service workloads to a storage account that is not locked with a **ReadOnly** lock.

## Next steps

- [Data protection overview](#)
- [Lock resources to prevent changes](#)

# Enable and manage soft delete for containers

8/22/2022 • 3 minutes to read • [Edit Online](#)

Container soft delete protects your data from being accidentally or erroneously modified or deleted. When container soft delete is enabled for a storage account, a container and its contents may be recovered after it has been deleted, within a retention period that you specify. For more details about container soft delete, see [Soft delete for containers](#).

For end-to-end data protection, Microsoft recommends that you also enable soft delete for blobs and blob versioning. To learn how to also enable soft delete for blobs, see [Enable and manage soft delete for blobs](#). To learn how to enable blob versioning, see [Blob versioning](#).

## Enable container soft delete

You can enable or disable container soft delete for the storage account at any time by using the Azure portal, PowerShell, Azure CLI, or an Azure Resource Manager template. Microsoft recommends setting the retention period for container soft delete to a minimum of seven days.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Template](#)

To enable container soft delete for your storage account by using Azure portal, follow these steps:

1. In the [Azure portal](#), navigate to your storage account.
2. Locate the **Data protection** settings under **Data management**.
3. Select **Enable soft delete for containers**.
4. Specify a retention period between 1 and 365 days.
5. Save your changes.

Data protection provides options for recovering your data when it is erroneously modified or deleted.

**Recovery**

- Enable operational backup with Azure Backup (preview)
- Enable point-in-time restore for containers
- Enable soft delete for blobs

Soft delete enables you to recover blobs that were previously marked for deletion, including blobs that were overwritten. [Learn more](#)

Keep deleted blobs for (in days)

**Enable soft delete for containers**

Soft delete enables you to recover containers that were previously marked for deletion. [Learn more](#)

Keep deleted containers for (in days)

**Tracking**

- Enable versioning for blobs
- Enable blob change feed

**Save** **Discard**

## View soft-deleted containers

When soft delete is enabled, you can view soft-deleted containers in the Azure portal. Soft-deleted containers are visible during the specified retention period. After the retention period expires, a soft-deleted container is permanently deleted and is no longer visible.

To view soft-deleted containers in the Azure portal, follow these steps:

1. Navigate to your storage account in the Azure portal and view the list of your containers.
2. Toggle the Show deleted containers switch to include deleted containers in the list.

Name	Status	Last modified	Public access level	Lease state
sample-container-1	Active	7/21/2020, 3:23:55 PM	Private	Available
sample-container-2	Deleted	7/21/2020, 3:23:55 PM	Private	-
sample-container-3	Deleted	7/21/2020, 3:23:55 PM	Private	-
sample-container-4	Active	7/21/2020, 3:23:55 PM	Private	Available
sample-container-5	Active	7/21/2020, 3:23:55 PM	Private	Available
sample-container-6	Active	7/21/2020, 3:23:55 PM	Private	Available

## Restore a soft-deleted container

You can restore a soft-deleted container and its contents within the retention period. To restore a soft-deleted container in the Azure portal, follow these steps:

1. Navigate to your storage account in the Azure portal and view the list of your containers.
2. Display the context menu for the container you wish to restore, and choose **Undelete** from the menu.

The screenshot shows the Azure Storage Explorer interface for a storage account named 'softdeletesamples'. The 'Containers' blade is active. A context menu is open over the second row, which contains the container 'sample-container-2'. The menu items shown are 'Container properties' and 'Undelete'. The container 'sample-container-2' is listed as 'Deleted'.

Name	Status	Last modified	Public access level	Lease state
sample-container-1	Active	7/21/2020, 3:23:55 PM	Private	Available
sample-container-2	Deleted	7/21/2020, 3:23:55 PM	Private	
sample-container-3	Deleted	7/21/2020, 3:23:55 PM	Private	
sample-container-4	Active	7/21/2020, 3:23:55 PM	Private	
sample-container-5	Active	7/21/2020, 3:23:55 PM	Private	Available
sample-container-6	Active	7/21/2020, 3:23:55 PM	Private	Available

## Next steps

- [Soft delete for containers](#)
- [Soft delete for blobs](#)
- [Blob versioning](#)

# Enable and manage blob versioning

8/22/2022 • 4 minutes to read • [Edit Online](#)

You can enable Blob storage versioning to automatically maintain previous versions of a blob when it is modified or deleted. When blob versioning is enabled, then you can restore an earlier version of a blob to recover your data if it is erroneously modified or deleted.

This article shows how to enable or disable blob versioning for the storage account by using the Azure portal or an Azure Resource Manager template. To learn more about blob versioning, see [Blob versioning](#).

## Enable blob versioning

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Template](#)

To enable blob versioning for a storage account in the Azure portal:

1. Navigate to your storage account in the portal.
2. Under **Blob service**, choose **Data protection**.
3. In the **Versioning** section, select **Enabled**.

The screenshot shows the Azure Storage portal interface for managing a storage account named "storagesamplesversioning". The "Data protection" section is currently selected. In the "Versioning" part of the configuration, the option "Turn on versioning for blobs" is checked and highlighted with a red box, indicating it is the active setting. Other options like "Turn on point-in-time restore for containers" and "Turn on soft delete" are also listed but not checked.

## Modify a blob to trigger a new version

The following code example shows how to trigger the creation of a new version with the Azure Storage client library for .NET, version [12.5.1](#) or later. Before running this example, make sure you have enabled versioning for your storage account.

The example creates a block blob, and then updates the blob's metadata. Updating the blob's metadata triggers the creation of a new version. The example retrieves the initial version and the current version, and shows that only the current version includes the metadata.

```
public static async Task UpdateVersionedBlobMetadata(BlobContainerClient blobContainerClient,
 string blobName)
{
 try
 {
 // Create the container.
 await blobContainerClient.CreateIfNotExistsAsync();

 // Upload a block blob.
 BlockBlobClient blockBlobClient = blobContainerClient.GetBlockBlobClient(blobName);

 string blobContents = string.Format("Block blob created at {0}.", DateTime.Now);
 byte[] byteArray = Encoding.ASCII.GetBytes(blobContents);

 string initialVersionId;
 using (MemoryStream stream = new MemoryStream(byteArray))
 {
 Response<BlobContentInfo> uploadResponse =
 await blockBlobClient.UploadAsync(stream, null, default);

 // Get the version ID for the current version.
 initialVersionId = uploadResponse.Value.VersionId;
 }

 // Update the blob's metadata to trigger the creation of a new version.
 Dictionary<string, string> metadata = new Dictionary<string, string>
 {
 { "key", "value" },
 { "key1", "value1" }
 };

 Response<BlobInfo> metadataResponse =
 await blockBlobClient.SetMetadataAsync(metadata);

 // Get the version ID for the new current version.
 string newVersionId = metadataResponse.Value.VersionId;

 // Request metadata on the previous version.
 BlockBlobClient initialVersionBlob = blockBlobClient.WithVersion(initialVersionId);
 Response<BlobProperties> propertiesResponse = await initialVersionBlob.GetPropertiesAsync();
 PrintMetadata(propertiesResponse);

 // Request metadata on the current version.
 BlockBlobClient newVersionBlob = blockBlobClient.WithVersion(newVersionId);
 Response<BlobProperties> newPropertiesResponse = await newVersionBlob.GetPropertiesAsync();
 PrintMetadata(newPropertiesResponse);
 }
 catch (RequestFailedException e)
 {
 Console.WriteLine(e.Message);
 Console.ReadLine();
 throw;
 }
}

static void PrintMetadata(Response<BlobProperties> propertiesResponse)
{
 if (propertiesResponse.Value.Metadata.Count > 0)
 {
 Console.WriteLine("Metadata values for version {0}:", propertiesResponse.Value.VersionId);
 foreach (var item in propertiesResponse.Value.Metadata)
 {
 Console.WriteLine("Key:{0} Value:{1}", item.Key, item.Value);
 }
 }
}
```

```

 }
 }
 else
 {
 Console.WriteLine("Version {0} has no metadata.", propertiesResponse.Value.VersionId);
 }
}

```

## List blob versions

To list blob versions or snapshots with the .NET v12 client library, specify the [BlobStates](#) parameter with the **Version** field.

The following code example shows how to list blobs version with the Azure Storage client library for .NET, version [12.5.1](#) or later. Before running this example, make sure you have enabled versioning for your storage account.

```

private static void ListBlobVersions(BlobContainerClient blobContainerClient,
 string blobName)
{
 try
 {
 // Call the listing operation, specifying that blob versions are returned.
 // Use the blob name as the prefix.
 var blobVersions = blobContainerClient.GetBlobs
 (BlobTraits.None, BlobStates.Version, prefix: blobName)
 .OrderByDescending(version => version.VersionId).Where(blob => blob.Name == blobName);

 // Construct the URI for each blob version.
 foreach (var version in blobVersions)
 {
 BlobUriBuilder blobUriBuilder = new BlobUriBuilder(blobContainerClient.Uri)
 {
 BlobName = version.Name,
 VersionId = version.VersionId
 };

 if ((bool)version.IsLatestVersion.GetValueOrDefault())
 {
 Console.WriteLine("Current version: {0}", blobUriBuilder);
 }
 else
 {
 Console.WriteLine("Previous version: {0}", blobUriBuilder);
 }
 }
 }
 catch (RequestFailedException e)
 {
 Console.WriteLine(e.Message);
 Console.ReadLine();
 throw;
 }
}

```

## Next steps

- [Blob versioning](#)
- [Soft delete for Azure Storage blobs](#)

# Enable soft delete for blobs

8/22/2022 • 4 minutes to read • [Edit Online](#)

Blob soft delete protects an individual blob and its versions, snapshots, and metadata 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 the blob to its state at deletion. After the retention period has expired, the blob is permanently deleted. For more information about blob soft delete, see [Soft delete for blobs](#).

Blob soft delete is part of a comprehensive data protection strategy for blob data. To learn more about Microsoft's recommendations for data protection, see [Data protection overview](#).

## Enable blob soft delete

You can enable or disable soft delete for a storage account at any time by using the Azure portal, PowerShell, or Azure CLI.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

Blob soft delete is enabled by default when you create a new storage account with the Azure portal. The setting to enable or disable blob soft delete when you create a new storage account is on the **Data protection** tab. For more information about creating a storage account, see [Create a storage account](#).

To enable blob soft delete for an existing storage account by using the Azure portal, follow these steps:

1. In the [Azure portal](#), navigate to your storage account.
2. Locate the **Data Protection** option under **Data management**.
3. In the **Recovery** section, select **Turn on soft delete for blobs**.
4. Specify a retention period between 1 and 365 days. Microsoft recommends a minimum retention period of seven days.
5. Save your changes.

The screenshot shows the Microsoft Azure portal interface for a storage account named 'contoso'. The left sidebar lists various management options like Data management, Geo-replication, and Data protection. The 'Data protection' option is selected and highlighted with a red box. The main content area is titled 'Recovery' and contains several configuration options. One option, 'Enable soft delete for blobs', is checked and highlighted with a red box. A tooltip for this option states: 'Soft delete enables you to recover blobs that were previously marked for deletion, including blobs that were overwritten.' Below this, there is a field to 'Keep deleted blobs for (in days)' with a value of '7'.

## Enable blob soft delete (hierarchical namespace)

Blob soft delete can also protect blobs and directories in accounts that have the hierarchical namespace feature enabled on them.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To enable blob soft delete for your storage account by using the Azure portal, follow these steps:

1. In the [Azure portal](#), navigate to your storage account.
2. Locate the **Data Protection** option under **Data Management**.
3. In the **Recovery** section, select **Enable soft delete for blobs**.
4. Specify a retention period between 1 and 365 days. Microsoft recommends a minimum retention period of seven days.
5. Save your changes.

Home &gt; contoso &gt; contoso

 contoso | Data protection X

Storage account

Search (Ctrl+ /) «

**Data management**

-  Geo-replication
-  **Data protection**
-  Blob inventory (preview)
-  Static website
-  Lifecycle management

**Settings**

-  Configuration
-  Resource sharing (CORS)
-  Advisor recommendations
-  Endpoints
-  Locks

Data protection provides options for recovering your data when it is erroneously modified or deleted.

**Recovery**

Enable soft delete for blobs

Soft delete enables you to recover blobs that were previously marked for deletion. For hierarchical namespace accounts, directories can be recovered. [Learn more](#)

Keep deleted blobs for (in days)

Enable soft delete for containers

## Next steps

- [Soft delete for blobs](#)
- [Manage and restore soft-deleted blobs](#)

# Manage and restore soft-deleted blobs

8/22/2022 • 11 minutes to read • [Edit Online](#)

Blob soft delete protects an individual blob and its versions, snapshots, and metadata 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 the blob to its state at deletion. After the retention period has expired, the blob is permanently deleted. For more information about blob soft delete, see [Soft delete for blobs](#).

Blob soft delete is part of a comprehensive data protection strategy for blob data. To learn more about Microsoft's recommendations for data protection, see [Data protection overview](#).

## Manage soft-deleted blobs

### Manage soft-deleted blobs with the Azure portal

You can use the Azure portal to view and restore soft-deleted blobs and snapshots.

#### View deleted blobs

When blobs are soft-deleted, they are invisible in the Azure portal by default. To view soft-deleted blobs, navigate to the **Overview** page for the container and toggle the **Show deleted blobs** setting. Soft-deleted blobs are displayed with a status of **Deleted**.

Authentication method: Access key ([Switch to Azure AD User Account](#))  
Location: container1



Name	Status	Modified	Access tier	Blob type	Size	Lease state	...
<input type="checkbox"/> blob1.txt	Deleted	3/26/2021, 12:19:04 ...	Hot (Inferred)	Block blob	48 B	-	...
<input type="checkbox"/> blob2.txt	Active	3/26/2021, 12:19:05 ...	Hot (Inferred)	Block blob	20 B	Available	...
<input type="checkbox"/> blob3.txt	Active	3/26/2021, 12:19:04 ...	Hot (Inferred)	Block blob	9 B	Available	...
<input type="checkbox"/> blob4.txt	Active	3/26/2021, 12:19:05 ...	Hot (Inferred)	Block blob	9 B	Available	...
<input type="checkbox"/> blob5.txt	Active	3/26/2021, 12:19:05 ...	Hot (Inferred)	Block blob	9 B	Available	...
<input type="checkbox"/> blob6.txt	Active	3/26/2021, 12:19:05 ...	Hot (Inferred)	Block blob	9 B	Available	...
<input type="checkbox"/> blob7.txt	Active	3/26/2021, 12:19:05 ...	Hot (Inferred)	Block blob	9 B	Available	...
<input type="checkbox"/> blob8.txt	Active	3/26/2021, 12:19:05 ...	Hot (Inferred)	Block blob	9 B	Available	...
<input type="checkbox"/> blob9.txt	Deleted	3/26/2021, 12:19:04 ...	Hot (Inferred)	Block blob	9 B	-	...

Next, select the deleted blob from the list of blobs to display its properties. Under the **Overview** tab, notice that the blob's status is set to **Deleted**. The portal also displays the number of days until the blob is permanently deleted.

## blob1.txt ...

Blob (deleted)

Refresh

[Overview](#) [Versions](#) [Snapshots](#)

### Properties

NAME	blob1.txt
TYPE	Block blob
SIZE	14 B
SERVER ENCRYPTED	true
ARCHIVE STATUS	-
REHYDRATE PRIORITY	-
ETAG	0x8DA69D2BBC28719
CONTENT-MD5	zhFORQHS9OLc6j4XtUbzOQ==
LEASE STATUS	-
LEASE STATE	-
LEASE DURATION	-
STATUS	Deleted
DELETION ID	
DELETED TIME	7/19/2022, 3:05:12 PM
REMAINING RETENTION DAYS	6

[Undelete](#)

### View deleted snapshots

Deleting a blob also deletes any snapshots associated with the blob. If a soft-deleted blob has snapshots, the deleted snapshots can also be displayed in the Azure portal. Display the soft-deleted blob's properties, then navigate to the **Snapshots** tab, and toggle **Show deleted snapshots**.

Dashboard > storagesamplesoftdelete > container1 >

## blob1.txt ...

Blob (deleted)

Refresh

[Overview](#) [Versions](#) [Snapshots](#) [Edit](#) [Generate SAS](#)

Filter snapshots

Show deleted snapshots

Name	Status	Modified	Access tier	Blob type	Content type	Size	...
<input type="checkbox"/> blob1.txt (3/26/2021, 4:3...  Deleted	Deleted	3/26/2021, 12:30:09 PM	Hot	Block blob	text/plain	48 B	...
<input type="checkbox"/> blob1.txt (3/26/2021, 4:3...  Deleted	Deleted	3/26/2021, 12:30:09 PM	Hot	Block blob	text/plain	48 B	...
<input type="checkbox"/> blob1.txt (3/26/2021, 4:3...  Deleted	Deleted	3/26/2021, 12:30:09 PM	Hot	Block blob	text/plain	48 B	...

### Restore soft-deleted objects when versioning is disabled

To restore a soft-deleted blob in the Azure portal when blob versioning is not enabled, first display the blob's properties, then select the **Undelete** button on the **Overview** tab. Restoring a blob also restores any snapshots that were deleted during the soft-delete retention period.

## blob1.txt ...

Blob (deleted)

Refresh

Overview Versions Snapshots

### Properties

NAME	blob1.txt
TYPE	Block blob
SIZE	14 B
SERVER ENCRYPTED	true
ARCHIVE STATUS	-
REHYDRATE PRIORITY	-
ETAG	0x8DA69D2BBC28719
CONTENT-MD5	zhFORQHS9OLc6j4XtUbzOQ==
LEASE STATUS	-
LEASE STATE	-
LEASE DURATION	-
STATUS	Deleted
DELETION ID	
DELETED TIME	7/19/2022, 3:05:12 PM
REMAINING RETENTION DAYS	6

Undelete

To promote a soft-deleted snapshot to the base blob, first make sure that the blob's soft-deleted snapshots have been restored. Select the **Undelete** button to restore the blob's soft-deleted snapshots, even if the base blob itself has not been soft-deleted. Next, select the snapshot to promote and use the **Promote snapshot** button to overwrite the base blob with the contents of the snapshot.

Dashboard > storagesamplessoftdelete > container1 >

## blob5.txt ...

Blob

Create snapshot Download snapshot Promote snapshot Refresh Delete snapshot Change tier

Overview Versions Snapshots Edit Generate SAS

Show deleted snapshots

Name	Status	Modified	Access tier	Blob type	Content type	Size	...
<input type="checkbox"/> blob5.txt (3/29/2021, 11:00....)	Active	3/26/2021, 12:30:09 PM	Hot	Block blob	text/plain	9 B	...
<input type="checkbox"/> blob5.txt (3/29/2021, 11:25....)	Active	3/26/2021, 12:30:09 PM	Hot	Block blob	text/plain	9 B	...
<input checked="" type="checkbox"/> blob5.txt (3/29/2021, 11:25....)	Active	3/29/2021, 11:25:25 AM	Hot	Block blob	text/plain	9 B	...
<input type="checkbox"/> blob5.txt (3/29/2021, 11:25....)	Active	3/29/2021, 11:25:39 AM	Hot	Block blob	text/plain	31 B	...
<input type="checkbox"/> blob5.txt (3/29/2021, 11:25....)	Active	3/29/2021, 11:25:43 AM	Hot	Block blob	text/plain	22 B	...
<input type="checkbox"/> blob5.txt (3/29/2021, 11:26....)	Active	3/29/2021, 11:25:46 AM	Hot	Block blob	text/plain	42 B	...

### Restore soft-deleted blobs when versioning is enabled

To restore a soft-deleted blob in the Azure portal when versioning is enabled, select the soft-deleted blob to display its properties, then select the **Versions** tab. Select the version that you want to promote to be the current version, then select **Make current version**.

Version Id	Status	Access tier	Blob type	Size
<input type="checkbox"/> 2021-03-26T16:19:04.954957Z	Active	Hot	Block blob	48 B
<input checked="" type="checkbox"/> 2021-03-26T16:25:18.7457679Z	Active	Hot	Block blob	48 B

To restore deleted versions or snapshots when versioning is enabled, display the blob's properties, then select the **Undelete** button on the **Overview** tab.

#### NOTE

When versioning is enabled, selecting the **Undelete** button on a deleted blob restores any soft-deleted versions or snapshots, but does not restore the base blob. To restore the base blob, you must promote a previous version.

## Manage soft-deleted blobs with code

You can use the Azure Storage client libraries to restore a soft-deleted blob or snapshot. The following examples show how to use the .NET client library.

### Restore soft-deleted objects when versioning is disabled

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

To restore deleted blobs when versioning is not enabled, call the [Undelete Blob](#) operation on those blobs. The **Undelete Blob** operation restores soft-deleted blobs and any deleted snapshots associated with those blobs.

Calling **Undelete Blob** on a blob that has not been deleted has no effect. The following example calls **Undelete Blob** on all blobs in a container, and restores the soft-deleted blobs and their snapshots:

```
foreach (BlobItem blob in container.GetBlobs(BlobTraits.None, BlobStates.Deleted))
{
 await container.GetBlockBlobClient(blob.Name).UndeleteAsync();
}
```

To restore a specific soft-deleted snapshot, first call the **Undelete Blob** operation on the base blob, then copy the desired snapshot over the base blob. The following example restores a block blob to the most recently generated snapshot:

```

// Restore the deleted blob.
await blockBlob.UndeleteAsync();

// List blobs in this container that match prefix.
// Include snapshots in listing.
Pageable<BlobItem> blobItems = container.GetBlobs
 (BlobTraits.None, BlobStates.Snapshots, prefix: blockBlob.Name);

// Get the URI for the most recent snapshot.
BlobUriBuilder blobSnapshotUri = new BlobUriBuilder(blockBlob.Uri)
{
 Snapshot = blobItems
 .OrderByDescending(snapshot => snapshot.Snapshot)
 .ElementAtOrDefault(0)?.Snapshot
};

// Restore the most recent snapshot by copying it to the blob.
blockBlob.StartCopyFromUri(blobSnapshotUri.ToUri());

```

#### **Restore soft-deleted blobs when versioning is enabled**

To restore a soft-deleted blob when versioning is enabled, copy a previous version over the base blob by using the [Copy Blob](#) or [Copy Blob From URL](#) operation.

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

```

// List blobs in this container that match prefix.
// Include versions in listing.
Pageable<BlobItem> blobItems = container.GetBlobs
 (BlobTraits.None, BlobStates.Version, prefix: blockBlob.Name);

// Get the URI for the most recent version.
BlobUriBuilder blobVersionUri = new BlobUriBuilder(blockBlob.Uri)
{
 VersionId = blobItems
 .OrderByDescending(version => version.VersionId)
 .ElementAtOrDefault(0)?.VersionId
};

// Restore the most recently generated version by copying it to the base blob.
blockBlob.StartCopyFromUri(blobVersionUri.ToUri());

```

## Manage soft-deleted blobs and directories (hierarchical namespace)

You can restore soft deleted blobs and directories in accounts that have a hierarchical namespace.

#### **Manage soft-deleted blobs with the Azure portal**

You can use the Azure portal to view and restore soft-deleted blobs and directories.

#### **View deleted blobs and directories**

When blobs or directories are soft-deleted, they are invisible in the Azure portal by default. To view soft-deleted blobs and directories, navigate to the [Overview](#) page for the container and toggle the **Show deleted blobs** setting. Soft-deleted blobs and directories are displayed with a status of **Deleted**. The following image shows a soft-deleted directory.

Authentication method: Access key ([Switch to Azure AD User Account](#))

Location: my-container

Search blobs by prefix (case-sensitive)  Show deleted blobs

Name	Status	Modified	Access tier	Blob type	Size	Lease state
<input type="checkbox"/> my-directory	Deleted				-	...
<input type="checkbox"/> my-directory-2	Active				-	...

#### NOTE

If you rename a directory that contains soft deleted items (subdirectories and blobs), those soft deleted items become disconnected from the directory, so they won't appear in the Azure portal when you toggle the **Show deleted blobs** setting. If you want to view them in the Azure portal, you'll have to revert the name of the directory back to its original name or create a separate directory that uses the original directory name.

Next, select the deleted directory or blob from the list display its properties. Under the **Overview** tab, notice that the status is set to **Deleted**. The portal also displays the number of days until the blob is permanently deleted.

## Directory properties ×

NAME	my-directory
LAST MODIFIED DATE	7/19/2022, 3:09:32 PM
ETAG	0x8DA69D35BD9E987
CONTENT TYPE	application/octet-stream
CONTENT ENCODING	-
CONTENT DISPOSITION	-
CACHE CONTROL	-
OWNER	-
PERMISSIONS	-
DELETION STATUS	Deleted
DELETION ID	133027421788273104
DELETION TIME	7/19/2022, 3:09:38 PM
REMAINING RETENTION DAYS	6

**Undelete**

#### Restore soft-delete blobs and directories

To restore a soft-deleted blob or directory in the Azure portal, first display the blob or directory's properties, then select the **Undelete** button on the **Overview** tab. The following image shows the Undelete button on a soft-deleted directory.

## Directory properties

X

NAME	my-directory
LAST MODIFIED DATE	7/19/2022, 3:09:32 PM
ETAG	0x8DA69D35BD9E987
CONTENT TYPE	application/octet-stream
CONTENT ENCODING	-
CONTENT DISPOSITION	-
CACHE CONTROL	-
OWNER	-
PERMISSIONS	-
DELETION STATUS	Deleted
DELETION ID	133027421788273104
DELETION TIME	7/19/2022, 3:09:38 PM
REMAINING RETENTION DAYS	6

[Undelete](#)

### Restore soft deleted blobs and directories by using PowerShell

#### IMPORTANT

This section applies only to accounts that have a hierarchical namespace.

1. Ensure that you have the **Az.Storage** preview module installed. For more information, see [Enable blob soft delete via PowerShell](#).
2. Obtain storage account authorization by using either a storage account key, a connection string, or Azure Active Directory (Azure AD). For more information, see [Connect to the account](#).

The following example obtains authorization by using a storage account key.

```
$ctx = New-AzStorageContext -StorageAccountName '<storage-account-name>' -StorageAccountKey
'<storage-account-key>'
```

3. To restore soft deleted item, use the `Restore-AzDataLakeGen2DeletedItem` command.

```
$filesystemName = "my-file-system"
$dirName="my-directory"
$deletedItems = Get-AzDataLakeGen2DeletedItem -Context $ctx -FileSystem $filesystemName -Path
$dirName
$deletedItems | Restore-AzDataLakeGen2DeletedItem
```

If you rename the directory that contains the soft deleted items, those items become disconnected from the directory. If you want to restore those items, you'll have to revert the name of the directory back to its original name or create a separate directory that uses the original directory name. Otherwise, you'll receive an error when you attempt to restore those soft deleted items.

### Restore soft deleted blobs and directories by using Azure CLI

#### IMPORTANT

This section applies only to accounts that have a hierarchical namespace.

1. Make sure that you have the `storage-preview` extension installed. For more information, see [Enable blob soft delete by using PowerShell](#).

2. Get a list of deleted items.

```
$filesystemName = "my-file-system"
az storage fs list-deleted-path -f $filesystemName --auth-mode login
```

3. To restore an item, use the `az storage fs undelete-path` command.

```
$dirName="my-directory"
az storage fs undelete-path -f $filesystemName --deleted-path-name $dirName --deletion-id "
<deletionId>" --auth-mode login
```

If you rename the directory that contains the soft deleted items, those items become disconnected from the directory. If you want to restore those items, you'll have to revert the name of the directory back to its original name or create a separate directory that uses the original directory name. Otherwise, you'll receive an error when you attempt to restore those soft deleted items.

## Restore soft deleted blobs and directories by using .NET

### IMPORTANT

This section applies only to accounts that have a hierarchical namespace.

1. Open a command prompt and change directory (`cd`) into your project folder For example:

```
cd myProject
```

2. Install the `Azure.Storage.Files.DataLake -v 12.7.0` version of the `Azure.Storage.Files.DataLake` NuGet package by using the `dotnet add package` command.

```
dotnet add package Azure.Storage.Files.DataLake -v -v 12.7.0 -s https://pkgs.dev.azure.com/azure-sdk/public/_packaging/azure-sdk-for-net/nuget/v3/index.json
```

3. Then, add these using statements to the top of your code file.

```
using Azure;
using Azure.Storage;
using Azure.Storage.Files.DataLake;
using Azure.Storage.Files.DataLake.Models;
using NUnit.Framework;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
```

4. The following code deletes a directory, and then restores a soft deleted directory.

This method assumes that you've created a `DataLakeServiceClient` instance. To learn how to create a `DataLakeServiceClient` instance, see [Connect to the account](#).

```

public void RestoreDirectory(DataLakeServiceClient serviceClient)
{
 DataLakeFileSystemClient fileSystemClient =
 serviceClient.GetFileSystemClient("my-container");

 DataLakeDirectoryClient directory =
 fileSystemClient.GetDirectoryClient("my-directory");

 // Delete the Directory
 await directory.DeleteAsync();

 // List Deleted Paths
 List<PathHierarchyDeletedItem> deletedItems = new List<PathHierarchyDeletedItem>();
 await foreach (PathHierarchyDeletedItem deletedItem in
fileSystemClient.GetDeletedPathsAsync())
 {
 deletedItems.Add(deletedItem);
 }

 Assert.AreEqual(1, deletedItems.Count);
 Assert.AreEqual("my-directory", deletedItems[0].Path.Name);
 Assert.IsTrue(deletedItems[0].IsPath);

 // Restore deleted directory.
 Response<DataLakePathClient> restoreResponse = await fileSystemClient.RestorePathAsync(
deletedItems[0].Path.Name,
deletedItems[0].Path.DeletionId);

}

```

If you rename the directory that contains the soft deleted items, those items become disconnected from the directory. If you want to restore those items, you'll have to revert the name of the directory back to its original name or create a separate directory that uses the original directory name. Otherwise, you'll receive an error when you attempt to restore those soft deleted items.

## Restore soft deleted blobs and directories by using Java

### IMPORTANT

This section applies only to accounts that have a hierarchical namespace.

- To get started, open the `pom.xml` file in your text editor. Add the following dependency element to the group of dependencies.

```

<dependency>
 <groupId>com.azure</groupId>
 <artifactId>azure-storage-file-datalake</artifactId>
 <version>12.6.0</version>
</dependency>

```

- Then, add these imports statements to your code file.

```
Put imports here
```

- The following snippet restores a soft deleted file named `my-file`.

This method assumes that you've created a `DataLakeServiceClient` instance. To learn how to create a `DataLakeServiceClient` instance, see [Connect to the account](#).

```
public void RestoreFile(DataLakeServiceClient serviceClient){

 DataLakeFileSystemClient fileSystemClient =
 serviceClient.getFileSystemClient("my-container");

 DataLakeFileClient fileClient =
 fileSystemClient.getFileClient("my-file");

 String deletionId = null;

 for (PathDeletedItem item : fileSystemClient.listDeletedPaths()) {

 if (item.getName().equals(fileClient.getFilePath())) {
 deletionId = item.getDeletionId();
 }
 }

 fileSystemClient.restorePath(fileClient.getFilePath(), deletionId);
}
```

If you rename the directory that contains the soft deleted items, those items become disconnected from the directory. If you want to restore those items, you'll have to revert the name of the directory back to its original name or create a separate directory that uses the original directory name. Otherwise, you'll receive an error when you attempt to restore those soft deleted items.

## Restore soft deleted blobs and directories by using Python

### IMPORTANT

This section applies only to accounts that have a hierarchical namespace.

1. Install version `12.4.0` or higher of the Azure Data Lake Storage client library for Python by using [pip](#).  
This command installs the latest version of the Azure Data Lake Storage client library for Python.

```
pip install azure-storage-file-datalake
```

2. Add these import statements to the top of your code file.

```
import os, uuid, sys
from azure.storage.filedatalake import DataLakeServiceClient
from azure.storage.filedatalake import FileSystemClient
```

3. The following code deletes a directory, and then restores a soft deleted directory.

The code example below contains an object named `service_client` of type `DataLakeServiceClient`. To see examples of how to create a `DataLakeServiceClient` instance, see [Connect to the account](#).

```
def restoreDirectory():

 try:
 global file_system_client

 file_system_client = service_client.create_file_system(file_system="my-file-system")

 directory_path = 'my-directory'
 directory_client = file_system_client.create_directory(directory_path)
 resp = directory_client.delete_directory()

 restored_directory_client = file_system_client.undelete_path(directory_client,
resp['deletion_id'])
 props = restored_directory_client.get_directory_properties()

 print(props)

 except Exception as e:
 print(e)
```

If you rename the directory that contains the soft deleted items, those items become disconnected from the directory. If you want to restore those items, you'll have to revert the name of the directory back to its original name or create a separate directory that uses the original directory name. Otherwise, you'll receive an error when you attempt to restore those soft deleted items.

## Next steps

- [Soft delete for Blob storage](#)
- [Enable soft delete for blobs](#)
- [Blob versioning](#)

# Perform a point-in-time restore on block blob data

8/22/2022 • 13 minutes to read • [Edit Online](#)

You can use point-in-time restore to restore one or more sets of block blobs to a previous state. This article describes how to enable point-in-time restore for a storage account and how to perform a restore operation.

To learn more about point-in-time restore, see [Point-in-time restore for block blobs](#).

**Caution**

Point-in-time restore supports restoring operations on block blobs only. Operations on containers cannot be restored. If you delete a container from the storage account by calling the [Delete Container](#) operation, that container cannot be restored with a restore operation. Rather than deleting an entire container, delete individual blobs if you may want to restore them later. Also, Microsoft recommends enabling soft delete for containers and blobs to protect against accidental deletion. For more information, see [Soft delete for containers](#) and [Soft delete for blobs](#).

## Enable and configure point-in-time restore

Before you enable and configure point-in-time restore, enable its prerequisites for the storage account: soft delete, change feed, and blob versioning. For more information about enabling each of these features, see these articles:

- [Enable soft delete for blobs](#)
- [Enable and disable the change feed](#)
- [Enable and manage blob versioning](#)

**IMPORTANT**

Enabling soft delete, change feed, and blob versioning may result in additional charges. For more information, see [Soft delete for blobs](#), [Change feed support in Azure Blob Storage](#), and [Blob versioning](#).

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To configure point-in-time restore with the Azure portal, follow these steps:

1. Navigate to your storage account in the Azure portal.
2. Under **Settings**, choose **Data Protection**.
3. Select **Turn on point-in-time restore**. When you select this option, soft delete for blobs, versioning, and change feed are also enabled.
4. Set the maximum restore point for point-in-time restore, in days. This number must be at least one day less than the retention period specified for blob soft delete.
5. Save your changes.

The following image shows a storage account configured for point-in-time restore with a restore point of seven days ago, and a retention period for blob soft delete of 14 days.

The screenshot shows the 'Data protection' settings for a storage account named 'dataprotectionsamples'. The 'Recovery' section is highlighted with a red box. It contains the following configuration:

- Turn on point-in-time restore: A note says to use point-in-time restore to restore one or more containers to an earlier state. It requires versioning, change feed, and blob soft delete to be enabled. A link to 'Learn more' is provided.
- Set the maximum restore point (days ago): A dropdown menu shows '7'.
- Turn on soft delete for blobs: A note says soft delete enables recovering blobs marked for deletion, including overwritten ones. A link to 'Learn more' is provided.
- Keep deleted blobs for (in days): A dropdown menu shows '14'.
- Turn on soft delete for containers: A note says soft delete enables recovering containers marked for deletion. A link to 'Learn more' is provided.
- Turn on versioning: A note says versioning maintains previous blob versions for recovery. A link to 'Learn more' is provided.
- Turn on blob change feed: A note says it tracks blob create, modification, and delete changes. A link to 'Learn more' is provided.

## Choose a restore point

The restore point is the date and time to which the data is restored. Azure Storage always uses a UTC date/time value as the restore point. However, the Azure portal allows you to specify the restore point in local time, and then converts that date/time value to a UTC date/time value to perform the restore operation.

When you perform a restore operation with PowerShell or Azure CLI, you should specify the restore point as a UTC date/time value. If the restore point is specified with a local time value instead of a UTC time value, the restore operation may still behave as expected in some cases. For example, if your local time is UTC minus five hours, then specifying a local time value results in a restore point that is five hours earlier than the value you provided. If no changes were made to the data in the range to be restored during that five-hour period, then the restore operation will produce the same results regardless of which time value was provided. Specifying a UTC time for the restore point is recommended to avoid unexpected results.

## Perform a restore operation

You can restore all containers in the storage account, or you can restore a range of blobs in one or more containers. A range of blobs is defined lexicographically, meaning in dictionary order. Up to ten lexicographical ranges are supported per restore operation. The start of the range is inclusive, and the end of the range is exclusive.

The container pattern specified for the start range and end range must include a minimum of three characters. The forward slash (/) that is used to separate a container name from a blob name does not count toward this minimum. A few examples for how to structure your restore ranges:

- To include the entire container named *myContainer* in the range for a restore use start range *myContainer* and end range *myContainer-0*. This shows how adding '-0' as a suffix to the container name for the end range value includes everything in the container for the restore.
- To include an entire virtual directory hierarchy, such as directory *myFolder* inside container *myContainer*, use start range *myContainer/myFolder/* and end range *myContainer/myFolder0*. Adding '0' as a suffix to virtual directory names for the end range includes all files with a prefix '*myContainer/myFolder/*' for the restore.

Wildcard characters are not supported in a lexicographical range. Any wildcard characters are treated as

standard characters.

You can restore blobs in the `$root` and `$web` containers by explicitly specifying them in a range passed to a restore operation. The `$root` and `$web` containers are restored only if they are explicitly specified. Other system containers cannot be restored.

Only block blobs are restored. Page blobs and append blobs are not included in a restore operation. For more information about limitations related to append blobs, see [Point-in-time restore for block blobs](#).

#### IMPORTANT

When you perform a restore operation, Azure Storage blocks data operations on the blobs in the ranges being restored for the duration of the operation. Read, write, and delete operations are blocked in the primary location. For this reason, operations such as listing containers in the Azure portal may not perform as expected while the restore operation is underway.

Read operations from the secondary location may proceed during the restore operation if the storage account is geo-replicated.

The time that it takes to restore a set of data is based on the number of write and delete operations made during the restore period after up to one hour for the restore job to be picked up. For example, an account with one million objects with 3,000 objects added per day and 1,000 objects deleted per day will require approximately two-three hours to restore to a point 30 days in the past. A restore with a small number of changes would require up to one hour to restore. A retention period and restoration more than 90 days in the past would not be recommended for an account with this rate of change.

### Restore all containers in the account

You can restore all containers in the storage account to return them to their previous state at a given point in time.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To restore all containers and blobs in the storage account with the Azure portal, follow these steps:

1. Navigate to the list of containers for your storage account.
2. On the toolbar, choose **Restore containers**, then **Restore all**.
3. In the **Restore all containers** pane, specify the restore point by providing a date and time.
4. Confirm that you want to proceed by checking the box.
5. Select **Restore** to begin the restore operation.

The screenshot shows the Azure portal interface for a storage account named 'dataprotectionsamples'. On the left, the 'Containers' blade is open, displaying a list of containers: \$blobchangefeed, Slogs, container1, container2, container3, container4, container5, container6, container7, and container8. Each container has a checkbox next to its name. On the right, a modal window titled 'Restore all containers' is displayed. It contains a 'Roll back to' dropdown set to '09/11/2020 11:15:00 AM'. Below the dropdown is a confirmation message: 'If you perform a restore, this action can't be undone. Are you sure you want to proceed?'. A checkbox labeled 'Yes, I want to proceed' is checked. A note below the checkbox states: 'Point-in-time restores support only block blobs, not page or append blobs. While the restore is happening, containers and blobs cannot be read or written to.' At the bottom of the modal are two buttons: 'Restore' (highlighted in blue) and 'Cancel'.

## Restore ranges of block blobs

You can restore one or more lexicographical ranges of blobs within a single container or across multiple containers to return those blobs to their previous state at a given point in time.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To restore a range of blobs in one or more containers with the Azure portal, follow these steps:

1. Navigate to the list of containers for your storage account.
2. Select the container or containers to restore.
3. On the toolbar, choose **Restore containers**, then **Restore selected**.
4. In the **Restore selected containers** pane, specify the restore point by providing a date and time.
5. Specify the ranges to restore. Use a forward slash (/) to delineate the container name from the blob prefix.
6. By default the **Restore selected containers** pane specifies a range that includes all blobs in the container. Delete this range if you do not want to restore the entire container. The default range is shown in the following image.



7. Confirm that you want to proceed by checking the box.
8. Select **Restore** to begin the restore operation.

The following image shows a restore operation on a set of ranges.

## Restore selected containers

X

Roll back to \* ⓘ

09/11/2020



11:15:00 AM

Blob ranges ⓘ

container1

to container1-0



container2/blob1

to container2/blob5



container3

to container5



e.g. myContainer/blobA

to e.g. myContainer/blobZ

If you perform a restore, this action can't be undone. Are you sure you want to proceed?

Yes, I want to proceed

ⓘ Point-in-time restores support only block blobs, not page or append blobs. While the restore is happening, containers and blobs cannot be read or written to.

**Restore**

Cancel

The restore operation shown in the image performs the following actions:

- Restores the complete contents of *container1*.
- Restores blobs in the lexicographical range *blob1* through *blob5* in *container2*. This range restores blobs with names such as *blob1*, *blob11*, *blob100*, *blob2*, and so on. Because the end of the range is exclusive, it restores blobs whose names begin with *blob4*, but does not restore blobs whose names begin with *blob5*.
- Restores all blobs in *container3* and *container4*. Because the end of the range is exclusive, this range does not restore *container5*.

## Next steps

- [Point-in-time restore for block blobs](#)
- [Soft delete](#)
- [Change feed](#)
- [Blob versioning](#)

# Create and manage a blob snapshot in .NET

8/22/2022 • 3 minutes to read • [Edit Online](#)

A snapshot is a read-only version of a blob that's taken at a point in time. This article shows how to create and manage blob snapshots using the [Azure Storage client library for .NET](#).

For more information about blob snapshots in Azure Storage, see [Blob snapshots](#).

## Create a snapshot

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

To create a snapshot of a block blob using version 12.x of the Azure Storage client library for .NET, use one of the following methods:

- [CreateSnapshot](#)
- [CreateSnapshotAsync](#)

The following code example shows how to create a snapshot with version 12.x. Include a reference to the [Azure.Identity](#) library to use your Azure AD credentials to authorize requests to the service. For more information about using the [DefaultAzureCredential](#) class to authorize a managed identity to access Azure Storage, see [Azure Identity client library for .NET](#).

```

private static async Task CreateBlockBlobSnapshot(string accountName, string containerName, string blobName,
Stream data)
{
 const string blobServiceEndpointSuffix = ".blob.core.windows.net";
 Uri containerUri = new Uri("https://" + accountName + blobServiceEndpointSuffix + "/" + containerName);

 // Get a container client object and create the container.
 BlobContainerClient containerClient = new BlobContainerClient(containerUri,
 new DefaultAzureCredential());
 await containerClient.CreateIfNotExistsAsync();

 // Get a blob client object.
 BlobClient blobClient = containerClient.GetBlobClient(blobName);

 try
 {
 // Upload text to create a block blob.
 await blobClient.UploadAsync(data);

 // Add blob metadata.
 IDictionary<string, string> metadata = new Dictionary<string, string>
 {
 { "ApproxBlobCreatedDate", DateTime.UtcNow.ToString() },
 { "FileType", "text" }
 };
 await blobClient.SetMetadataAsync(metadata);

 // Sleep 5 seconds.
 System.Threading.Thread.Sleep(5000);

 // Create a snapshot of the base blob.
 // You can specify metadata at the time that the snapshot is created.
 // If no metadata is specified, then the blob's metadata is copied to the snapshot.
 await blobClient.CreateSnapshotAsync();
 }
 catch (RequestFailedException e)
 {
 Console.WriteLine(e.Message);
 Console.ReadLine();
 throw;
 }
}
}

```

## Delete snapshots

To delete a blob, you must first delete any snapshots of that blob. You can delete a snapshot individually, or specify that all snapshots be deleted when the source blob is deleted. If you attempt to delete a blob that still has snapshots, an error results.

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

To delete a blob and its snapshots using version 12.x of the Azure Storage client library for .NET, use one of the following methods, and include the [DeleteSnapshotsOption](#) enum:

- [Delete](#)
- [DeleteAsync](#)
- [DeleteIfExists](#)
- [DeleteIfExistsAsync](#)

The following code example shows how to delete a blob and its snapshots in .NET, where `blobClient` is an

object of type [BlobClient](#)):

```
await blobClient.DeleteIfExistsAsync(DeleteSnapshotsOption.IncludeSnapshots, null, default);
```

## Next steps

- [Blob snapshots](#)
- [Blob versions](#)
- [Soft delete for blobs](#)

# Process change feed in Azure Blob Storage

8/22/2022 • 5 minutes to read • [Edit Online](#)

Change feed provides transaction logs of all the changes that occur to the blobs and the blob metadata in your storage account. This article shows you how to read change feed records by using the blob change feed processor library.

To learn more about the change feed, see [Change feed in Azure Blob Storage](#).

## Get the blob change feed processor library

1. Open a command window (For example: Windows PowerShell).
2. From your project directory, install the [Azure.Storage.Blobs.Changefeed NuGet package](#).

```
dotnet add package Azure.Storage.Blobs --version 12.5.1
dotnet add package Azure.Storage.Blobs.ChangeFeed --version 12.0.0-preview.4
```

## Read records

### NOTE

The change feed is an immutable and read-only entity in your storage account. Any number of applications can read and process the change feed simultaneously and independently at their own convenience. Records aren't removed from the change feed when an application reads them. The read or iteration state of each consuming reader is independent and maintained by your application only.

This example iterates through all records in the change feed, adds them to a list, and then returns that list to the caller.

```
public async Task<List<BlobChangeFeedEvent>> ChangeFeedAsync(string connectionString)
{
 // Get a new blob service client.
 BlobServiceClient blobServiceClient = new BlobServiceClient(connectionString);

 // Get a new change feed client.
 BlobChangeFeedClient changeFeedClient = blobServiceClient.GetChangeFeedClient();

 List<BlobChangeFeedEvent> changeFeedEvents = new List<BlobChangeFeedEvent>();

 // Get all the events in the change feed.
 await foreach (BlobChangeFeedEvent changeFeedEvent in changeFeedClient.GetChangesAsync())
 {
 changeFeedEvents.Add(changeFeedEvent);
 }

 return changeFeedEvents;
}
```

This example prints to the console a few values from each record in the list.

```

public void showEventData(List<BlobChangeFeedEvent> changeFeedEvents)
{
 foreach (BlobChangeFeedEvent changeFeedEvent in changeFeedEvents)
 {
 string subject = changeFeedEvent.Subject;
 string eventType = changeFeedEvent.EventType.ToString();
 string api = changeFeedEvent.EventData.Api;

 Console.WriteLine("Subject: " + subject + "\n" +
 "Event Type: " + eventType + "\n" +
 "Api: " + api);
 }
}

```

## Resume reading records from a saved position

You can choose to save your read position in the change feed, and then resume iterating through the records at a future time. You can save the read position by getting the change feed cursor. The cursor is a **string** and your application can save that string in any way that makes sense for your application's design (For example: to a file, or database).

This example iterates through all records in the change feed, adds them to a list, and saves the cursor. The list and the cursor are returned to the caller.

```

public async Task<(string, List<BlobChangeFeedEvent>)> ChangeFeedResumeWithCursorAsync
 (string connectionString, string cursor)
{
 // Get a new blob service client.
 BlobServiceClient blobServiceClient = new BlobServiceClient(connectionString);

 // Get a new change feed client.
 BlobChangeFeedClient changeFeedClient = blobServiceClient.GetChangeFeedClient();
 List<BlobChangeFeedEvent> changeFeedEvents = new List<BlobChangeFeedEvent>();

 IAsyncEnumerator<Page<BlobChangeFeedEvent>> enumerator = changeFeedClient
 .GetChangesAsync(continuation: cursor)
 .AsPages(pageSizeHint: 10)
 .GetAsyncEnumerator();

 await enumerator.MoveNextAsync();

 foreach (BlobChangeFeedEvent changeFeedEvent in enumerator.Current.Values)
 {

 changeFeedEvents.Add(changeFeedEvent);
 }

 // Update the change feed cursor. The cursor is not required to get each page of events,
 // it is intended to be saved and used to resume iterating at a later date.
 cursor = enumerator.Current.ContinuationToken;
 return (cursor, changeFeedEvents);
}

```

## Stream processing of records

You can choose to process change feed records as they are committed to the change feed. See [Specifications](#). The change events are published to the change feed at a period of 60 seconds on average. We recommend that you poll for new changes with this period in mind when specifying your poll interval.

This example periodically polls for changes. If change records exist, this code processes those records and saves

change feed cursor. That way if the process is stopped and then started again, the application can use the cursor to resume processing records where it last left off. This example saves the cursor to a local application configuration file, but your application can save it in any form that makes the most sense for your scenario.

```
public async Task ChangeFeedStreamAsync
 (string connectionString, int waitTimeMs, string cursor)
{
 // Get a new blob service client.
 BlobServiceClient blobServiceClient = new BlobServiceClient(connectionString);

 // Get a new change feed client.
 BlobChangeFeedClient changeFeedClient = blobServiceClient.GetChangeFeedClient();

 while (true)
 {
 IAsyncEnumerator<Page<BlobChangeFeedEvent>> enumerator = changeFeedClient
 .GetChangesAsync(continuation: cursor).AsPages().GetAsyncEnumerator();

 while (true)
 {
 var result = await enumerator.MoveNextAsync();

 if (result)
 {
 foreach (BlobChangeFeedEvent changeFeedEvent in enumerator.Current.Values)
 {
 string subject = changeFeedEvent.Subject;
 string eventType = changeFeedEvent.EventType.ToString();
 string api = changeFeedEvent.EventData.Api;

 Console.WriteLine("Subject: " + subject + "\n" +
 "Event Type: " + eventType + "\n" +
 "Api: " + api);
 }

 // helper method to save cursor.
 SaveCursor(enumerator.Current.ContinuationToken);
 }
 else
 {
 break;
 }
 }

 await Task.Delay(waitTimeMs);
 }
}

public void SaveCursor(string cursor)
{
 System.Configuration.Configuration config =
 ConfigurationManager.OpenExeConfiguration
 (ConfigurationUserLevel.None);

 config.AppSettings.Settings.Clear();
 config.AppSettings.Settings.Add("Cursor", cursor);
 config.Save(ConfigurationSaveMode.Modified);
}
```

## Reading records within a time range

You can read records that fall within a specific time range. This example iterates through all records in the change feed that fall between 3:00 PM on March 2 2020 and 2:00 AM on August 7 2020, adds them to a list, and

then returns that list to the caller.

## Selecting segments for a time range

```
public async Task<List<BlobChangeFeedEvent>> ChangeFeedBetweenDatesAsync(string connectionString)
{
 // Get a new blob service client.
 BlobServiceClient blobServiceClient = new BlobServiceClient(connectionString);

 // Get a new change feed client.
 BlobChangeFeedClient changeFeedClient = blobServiceClient.GetChangeFeedClient();
 List<BlobChangeFeedEvent> changeFeedEvents = new List<BlobChangeFeedEvent>();

 // Create the start and end time. The change feed client will round start time down to
 // the nearest hour, and round endTime up to the next hour if you provide DateTimeOffsets
 // with minutes and seconds.
 DateTimeOffset startTime = new DateTimeOffset(2020, 3, 2, 15, 0, 0, TimeSpan.Zero);
 DateTimeOffset endTime = new DateTimeOffset(2020, 8, 7, 2, 0, 0, TimeSpan.Zero);

 // You can also provide just a start or end time.
 await foreach (BlobChangeFeedEvent changeFeedEvent in changeFeedClient.GetChangesAsync(
 start: startTime,
 end: endTime))
 {
 changeFeedEvents.Add(changeFeedEvent);
 }

 return changeFeedEvents;
}
```

The start time that you provide is rounded down to the nearest hour and the end time is rounded up to the nearest hour. It's possible that users might see events that occurred before the start time and after the end time. It's also possible that some events that occur between the start and end time won't appear. That's because events might be recorded during the hour previous to the start time or during the hour after the end time.

## Next steps

- [Data protection overview](#)
- [Change feed in Azure Blob Storage](#)

# Configure immutability policies for blob versions

8/22/2022 • 20 minutes to read • [Edit Online](#)

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. Immutability policies include time-based retention policies and legal holds. For more information about immutability policies for Blob Storage, see [Store business-critical blob data with immutable storage](#).

An immutability policy may be scoped either to an individual blob version or to a container. This article describes how to configure a version-level immutability policy. To learn how to configure container-level immutability policies, see [Configure immutability policies for containers](#).

Configuring a version-level immutability policy is a two-step process:

1. First, enable support for version-level immutability on a new storage account or on a new or existing container. See [Enable support for version-level immutability](#) for details.
2. Next, configure a time-based retention policy or legal hold that applies to one or more blob versions in that container.

## Prerequisites

To configure version-level time-based retention policies, blob versioning must be enabled for the storage account. Keep in mind that enabling blob versioning may have a billing impact. To learn how to enable blob versioning, see [Enable and manage blob versioning](#).

For information about supported storage account configurations for version-level immutability policies, see [Supported account configurations](#).

## Enable support for version-level immutability

Before you can apply a time-based retention policy to a blob version, you must enable support for version-level immutability. You can enable support for version-level immutability on a new storage account, or on a new or existing container.

### Enable version-level immutability support on a storage account

You can enable support for version-level immutability only when you create a new storage account.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To enable support for version-level immutability when you create a storage account in the Azure portal, follow these steps:

1. Navigate to the **Storage accounts** page in the Azure portal.
2. Select the **Create** button to create a new account.
3. Fill out the **Basics** tab.
4. On the **Data protection** tab, under Access control, select **Enable version-level immutability support**. When you check this box, the box for **Enable versioning for blobs** is also automatically

checked.

5. Select **Review + Create** to validate your account parameters and create the storage account.

Dashboard > Storage accounts >

## Create a storage account

Basics Advanced Networking Data protection Tags Review + create

### Recovery

Protect your data from accidental or erroneous deletion or modification.

Enable point-in-time restore for containers  
Use point-in-time restore to restore one or more containers to an earlier state. If point-in-time restore is enabled, then versioning, change feed, and blob soft delete must also be enabled. [Learn more](#)

Enable soft delete for blobs  
Soft delete enables you to recover blobs that were previously marked for deletion, including blobs that were overwritten. [Learn more](#)

Days to retain deleted blobs

Enable soft delete for containers  
Soft delete enables you to recover containers that were previously marked for deletion. [Learn more](#)

Days to retain deleted containers

Enable soft delete for file shares  
Soft delete enables you to recover file shares that were previously marked for deletion. [Learn more](#)

Days to retain deleted file shares

### Tracking

Manage versions and keep track of changes made to your blob data.

Enable versioning for blobs  
Use versioning to automatically maintain previous versions of your blobs for recovery and restoration. [Learn more](#)

Enable blob change feed  
Keep track of create, modification, and delete changes to blobs in your account. [Learn more](#)

**Access control**

Enable version-level immutability support  
Allows you to set time-based retention policies for blob versions. You can set a default policies at the account or container level, or set policies for specific blobs or versions. Versioning is required for this property to be enabled. [Learn more](#)

[Review + create](#) [< Previous](#) [Next : Tags >](#)

After the storage account is created, you can configure a default version-level policy for the account. For more information, see [Configure a default time-based retention policy](#).

If version-level immutability support is enabled for the storage account and the account contains one or more containers, then you must delete all containers before you delete the storage account, even if there are no immutability policies in effect for the account or containers.

#### Enable version-level immutability support on a container

Both new and existing containers can be configured to support version-level immutability. However, an existing container must undergo a migration process in order to enable support.

Keep in mind that enabling version-level immutability support for a container doesn't make data in that

container immutable. You must also either configure a default immutability policy for the container, or an immutability policy on a specific blob version. If you enabled version-level immutability for the storage account when it was created, you can also configure a default immutability policy for the account.

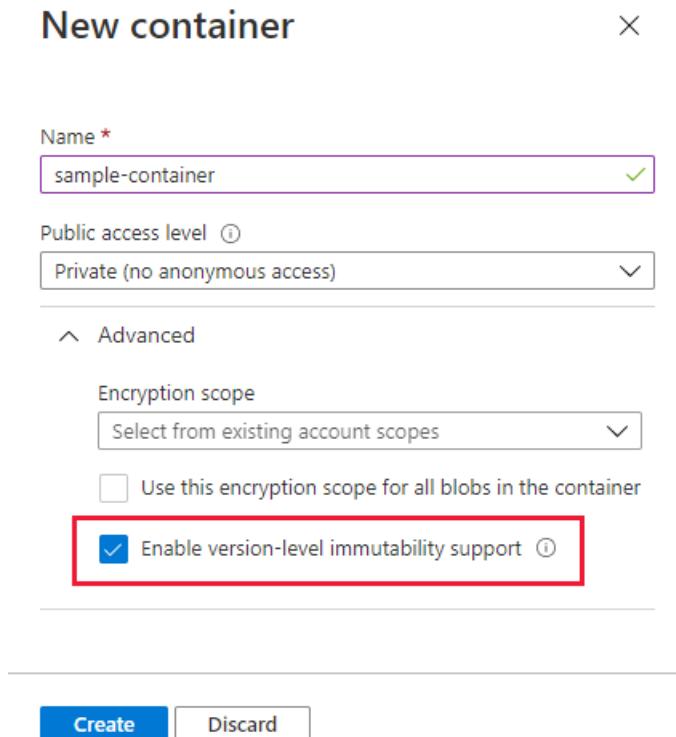
#### Enable version-level immutability for a new container

To use a version-level immutability policy, you must first explicitly enable support for version-level WORM on the container. You can enable support for version-level WORM either when you create the container, or when you add a version-level immutability policy to an existing container.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To create a container that supports version-level immutability in the Azure portal, follow these steps:

1. Navigate to the **Containers** page for your storage account in the Azure portal, and select **Add**.
2. In the **New container** dialog, provide a name for your container, then expand the **Advanced** section.
3. Select **Enable version-level immutability support** to enable version-level immutability for the container.



If version-level immutability support is enabled for a container and the container contains one or more blobs, then you must delete all blobs in the container before you can delete the container, even if there are no immutability policies in effect for the container or its blobs.

#### Migrate an existing container to support version-level immutability

To configure version-level immutability policies for an existing container, you must migrate the container to support version-level immutable storage. Container migration may take some time and can't be reversed. You can migrate 10 containers at a time per storage account.

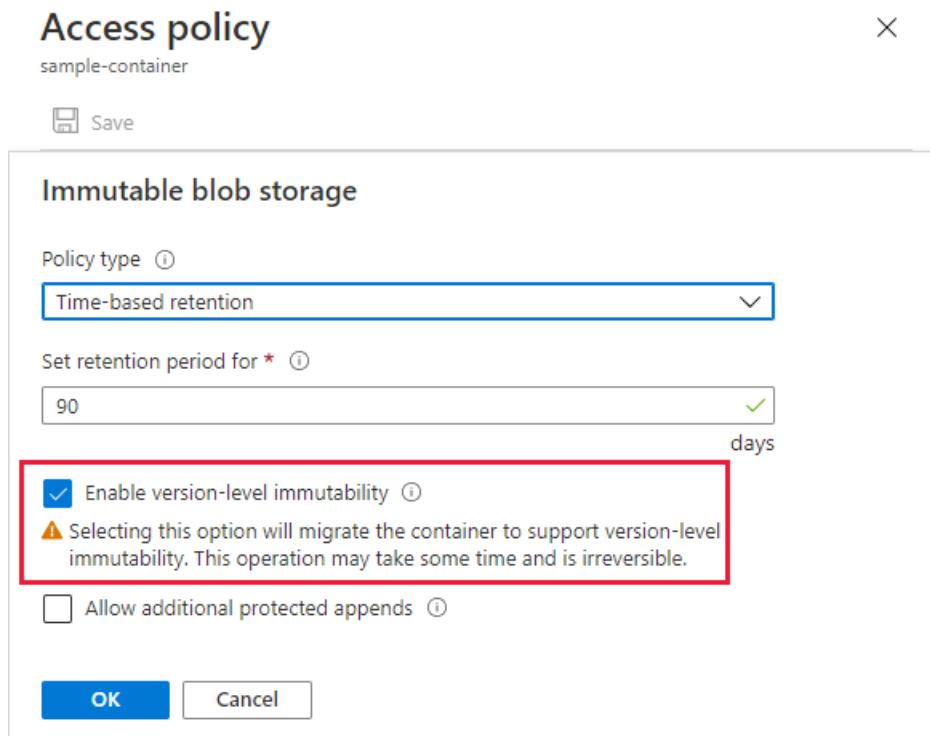
To migrate an existing container to support version-level immutability policies, the container must have a container-level time-based retention policy configured. The migration fails unless the container has an existing policy. The retention interval for the container-level policy is maintained as the retention interval for the default version-level policy on the container.

If the container has an existing container-level legal hold, then it can't be migrated until the legal hold is removed.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To migrate a container to support version-level immutability policies in the Azure portal, follow these steps:

1. Navigate to the desired container.
2. Select the **More** button on the right, then select **Access policy**.
3. Under **Immutable blob storage**, select **Add policy**.
4. For the **Policy type** field, choose *Time-based retention*, and specify the retention interval.
5. Select **Enable version-level immutability**.
6. Select **OK** to create a container-level policy with the specified retention interval and then begin the migration to version-level immutability support.



While the migration operation is underway, the scope of the policy on the container shows as *Container*. Any operations related to managing version-level immutability policies aren't permitted while the container migration is in progress. Other operations on blob data will proceed normally during migration.

## Access policy

sample-container

X



Save

**i** Your container is currently being migrated to support version-level immutability. The migration may take a while and certain operations on the container and on the blobs inside it may not be available until the migration is completed.

Stored access policies

Identifier	Start time	Expiry time	Permissions
No results			

**+** Add policy

Immutable blob storage (i)

Identifier	Scope	Retention interval	State
Time-based retention	Container	90 days	Unlocked

**+** Add policy

After the migration is complete, the scope of the policy on the container shows as *Version*. The policy shown is a default policy on the container that automatically applies to all blob versions subsequently created in the container. The default policy can be overridden on any version by specifying a custom policy for that version.

## Access policy

sample-container

X



Save

Stored access policies

Identifier	Start time	Expiry time	Permissions
No results			

**+** Add policy

Immutable blob storage (i)

Identifier	Scope	Retention interval	State
Time-based retention	Version	90 days	Unlocked

## Configure a default time-based retention policy

After you have enabled version-level immutability support for a storage account or for an individual container, you can specify a default version-level time-based retention policy for the account or container. When you specify a default policy for an account or container, that policy applies by default to all new blob versions that are created in the account or container. You can override the default policy for any individual blob version in the account or container.

The default policy isn't automatically applied to blob versions that existed before the default policy was

configured.

If you migrated an existing container to support version-level immutability, then the container-level policy that was in effect before the migration is migrated to a default version-level policy for the container.

To configure a default version-level immutability policy for a storage account or container, use the Azure portal, PowerShell, Azure CLI, or one of the Azure Storage SDKs. Make sure that you have enabled support for version-level immutability for the storage account or container, as described in [Enable support for version-level immutability](#).

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To configure a default version-level immutability policy for a storage account in the Azure portal, follow these steps:

1. In the Azure portal, navigate to your storage account.
2. Under **Data management**, select **Data protection**.
3. On the **Data protection** page, locate the **Access control** section. If the storage account was created with support for version-level immutability, then the **Manage policy** button appears in the **Access control** section.

Access control

Enable version-level immutability support

Allows you to set time-based retention policy on the account-level that will apply to all blob versions. Enable this feature to set a default policy at the account level. Without enabling this, you can still set a default policy at the container level or set policies for specific blob versions. This property can be enabled only during creation. [Learn more](#)

**Manage policy**

4. Select the **Manage policy** button to display the **Manage version-level immutability policy** dialog.
5. Add a default time-based retention policy for the storage account.

**Manage version-level immutability policy** ×

ⓘ A policy created here will be applied to every new blob version unless it is overwritten with a policy at the container or blob version level. [Learn more](#)

Identifier	Retention interval	State	...
Time-based retention	90 days	Unlocked	...

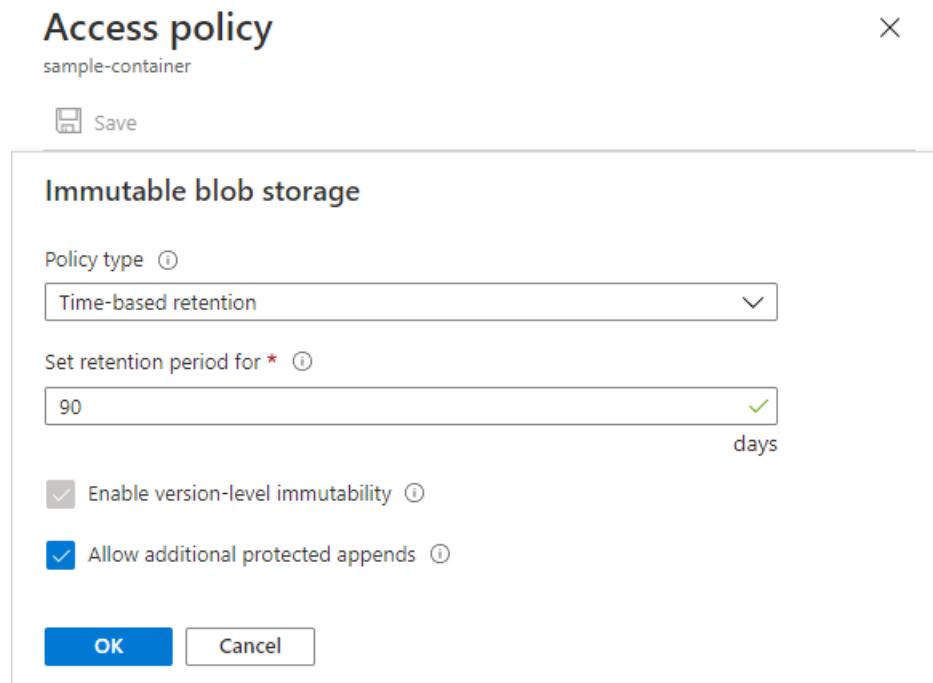
**Add policy**

To configure a default version-level immutability policy for a container in the Azure portal, follow these steps:

1. In the Azure portal, navigate to the **Containers** page, and locate the container to which you want to

apply the policy.

2. Select the **More** button to the right of the container name, and choose **Access policy**.
3. In the **Access policy** dialog, under the **Immutable blob storage** section, choose **Add policy**.
4. Select **Time-based retention policy** and specify the retention interval.
5. If desired, select **Allow additional protected appends** to enable writes to append blobs that are protected by an immutability policy. For more information, see [Allow protected append blobs writes](#).
6. Select **OK** to apply the default policy to the container.



#### Determine the scope of a retention policy on a container

To determine the scope of a time-based retention policy in the Azure portal, follow these steps:

1. Navigate to the desired container.
2. Select the **More** button on the right, then select **Access policy**.
3. Under **Immutable blob storage**, locate the **Scope** field. If the container is configured with a default version-level retention policy, then the scope is set to *Version*, as shown in the following image:

## Access policy

sample-container-vlw

X



Stored access policies

Identifier	Start time	Expiry time	Permissions
No results			

+ Add policy

Immutable blob storage ⓘ

Identifier	Scope	Retention interval	State
Time-based retention	Version	10 days	Unlocked

4. If the container is configured with a container-level retention policy, then the scope is set to *Container*, as shown in the following image:

## Access policy

sample-container-clw

X



Stored access policies

Identifier	Start time	Expiry time	Permissions
No results			

+ Add policy

Immutable blob storage ⓘ

Identifier	Scope	Retention interval	State
Time-based retention	Container	1 days	Unlocked

+ Add policy

## Configure a time-based retention policy on an existing version

Time-based retention policies maintain blob data in a WORM state for a specified interval. For more information about time-based retention policies, see [Time-based retention policies for immutable blob data](#).

You have three options for configuring a time-based retention policy for a blob version:

- Option 1: You can configure a default policy on the storage account or container that applies to all objects in the account or container. Objects in the account or container will inherit the default policy unless you explicitly override it by configuring a policy on an individual blob version. For more information, see [Configure a default time-based retention policy](#).
- Option 2: You can configure a policy on the current version of the blob. This policy can override a default policy configured on the storage account or container, if a default policy exists and it's unlocked. By default,

any previous versions that are created after the policy is configured will inherit the policy on the current version of the blob. For more information, see [Configure a retention policy on the current version of a blob](#).

- Option 3: You can configure a policy on a previous version of a blob. This policy can override a default policy configured on the current version, if one exists and it's unlocked. For more information, see [Configure a retention policy on a previous version of a blob](#).

For more information on blob versioning, see [Blob versioning](#).

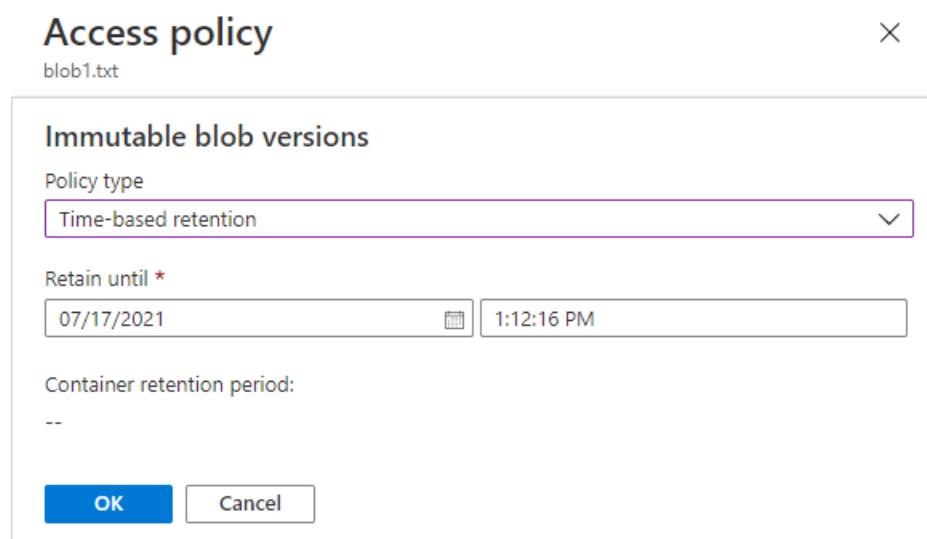
- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

The Azure portal displays a list of blobs when you navigate to a container. Each blob displayed represents the current version of the blob. You can access a list of previous versions by selecting the **More** button for a blob and choosing **View previous versions**.

### Configure a retention policy on the current version of a blob

To configure a time-based retention policy on the current version of a blob, follow these steps:

1. Navigate to the container that contains the target blob.
2. Select the **More** button to the right of the blob name, and choose **Access policy**. If a time-based retention policy has already been configured for the previous version, it appears in the **Access policy** dialog.
3. In the **Access policy** dialog, under the **Immutable blob versions** section, choose **Add policy**.
4. Select **Time-based retention policy** and specify the retention interval.
5. Select **OK** to apply the policy to the current version of the blob.



You can view the properties for a blob to see whether a policy is enabled on the current version. Select the blob, then navigate to the **Overview** tab and locate the **Version-level immutability policy** property. If a policy is enabled, the **Retention period** property will display the expiry date and time for the policy. Keep in mind that a policy may either be configured for the current version, or may be inherited from the blob's parent container if a default policy is in effect.

**blob1.txt** ... X

Blob

Save Discard Download Refresh Delete Change tier ...

Overview Versions Snapshots Edit Generate SAS

Properties

URL	<a href="https://immutableamps...">https://immutableamps...</a> 
LAST MODIFIED	6/22/2021, 4:49:40 PM
CREATION TIME	6/22/2021, 4:49:40 PM
VERSION ID	2021-06-22T23:49:40.3472021Z
TYPE	Block blob
SIZE	75 B
ACCESS TIER	Hot (Inferred)
ACCESS TIER LAST MODIFIED	N/A
SERVER ENCRYPTED	true
ETAG	0x8D935D866CB9585
VERSION-LEVEL IMMUTABILITY POLICY	Enabled
RETENTION PERIOD	7/17/2021, 10:26:18 PM
CONTENT-TYPE	text/plain
CONTENT-MD5	aBCi9jjbxTL7NWpVZSjr3Q==
LEASE STATUS	Unlocked
LEASE STATE	Available
LEASE DURATION	-
COPY STATUS	-
COPY COMPLETION TIME	-

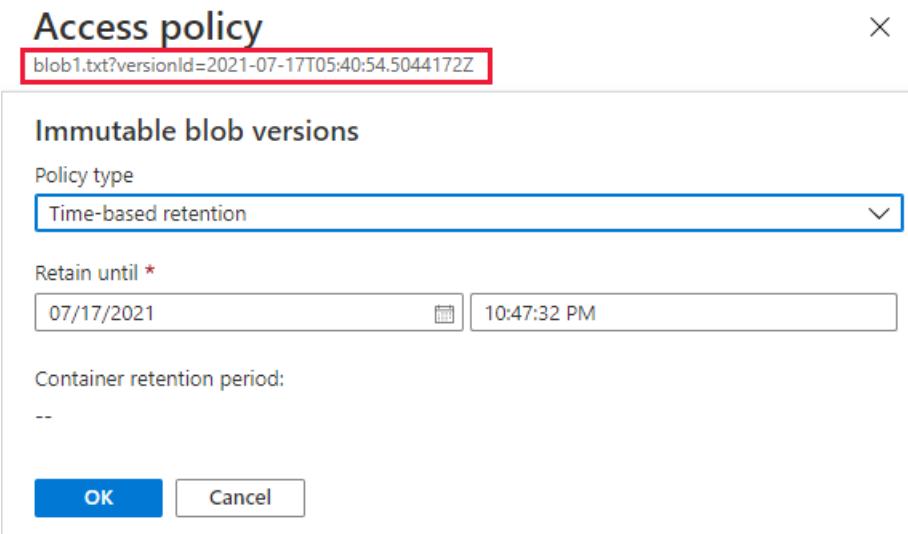
**Undelete**

### Configure a retention policy on a previous version of a blob

You can also configure a time-based retention policy on a previous version of a blob. A previous version is always immutable in that it can't be modified. However, a previous version can be deleted. A time-based retention policy protects against deletion while it is in effect.

To configure a time-based retention policy on a previous version of a blob, follow these steps:

1. Navigate to the container that contains the target blob.
2. Select the blob, then navigate to the **Versions** tab.
3. Locate the target version, then select the **More** button and choose **Access policy**. If a time-based retention policy has already been configured for the previous version, it appears in the **Access policy** dialog.
4. In the **Access policy** dialog, under the **Immutable blob versions** section, choose **Add policy**.
5. Select **Time-based retention policy** and specify the retention interval.
6. Select **OK** to apply the policy to the current version of the blob.



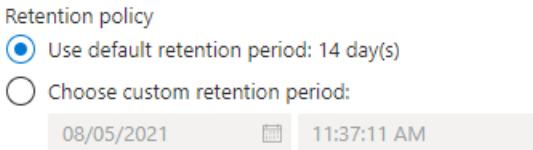
## Configure a time-based retention policy when uploading a blob

When you use the Azure portal to upload a blob to a container that supports version-level immutability, you have several options for configuring a time-based retention policy for the new blob:

- Option 1: If a default retention policy is configured for the container, you can upload the blob with the container's policy. This option is selected by default when there's a retention policy on the container.
- Option 2: If a default retention policy is configured for the container, you can choose to override the default policy, either by defining a custom retention policy for the new blob, or by uploading the blob with no policy.
- Option 3: If no default policy is configured for the container, then you can upload the blob with a custom policy, or with no policy.

To configure a time-based retention policy when you upload a blob, follow these steps:

1. Navigate to the desired container, and select **Upload**.
2. In the **Upload blob** dialog, expand the **Advanced** section.
3. Configure the time-based retention policy for the new blob in the **Retention policy** field. If there's a default policy configured for the container, that policy is selected by default. You can also specify a custom policy for the blob.



## Modify or delete an unlocked retention policy

You can modify an unlocked time-based retention policy to shorten or lengthen the retention interval. You can also delete an unlocked policy. Editing or deleting an unlocked time-based retention policy for a blob version doesn't affect policies in effect for any other versions. If there's a default time-based retention policy in effect for the container, then the blob version with the modified or deleted policy will no longer inherit from the container.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To modify an unlocked time-based retention policy in the Azure portal, follow these steps:

1. Locate the target container or version. Select the **More** button and choose **Access policy**.
2. Locate the existing unlocked immutability policy. Select the **More** button, then select **Edit** from the menu.

The screenshot shows the 'Access policy' blade for a blob named 'blob2.txt'. Under the 'Immutable blob versions' section, there is a table with one row. The row contains the identifier 'Time-based retention', the retention date '7/18/2021, 4:28:57 PM', and a 'State' column. A context menu is open over the 'Time-based retention' row, listing three options: 'Lock policy' (selected), 'Edit' (with a cursor icon pointing at it), and 'Delete'.

3. Provide the new date and time for the policy expiration.

To delete the unlocked policy, select **Delete** from the **More** menu.

## Lock a time-based retention policy

When you have finished testing a time-based retention policy, you can lock the policy. A locked policy is compliant with SEC 17a-4(f) and other regulatory compliance. You can lengthen the retention interval for a locked policy up to five times, but you can't shorten it.

After a policy is locked, you can't delete it. However, you can delete the blob after the retention interval has expired.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To lock a policy in the Azure portal, follow these steps:

1. Locate the target container or version. Select the **More** button and choose **Access policy**.
2. Under the **Immutable blob versions** section, locate the existing unlocked policy. Select the **More** button, then select **Lock policy** from the menu.
3. Confirm that you want to lock the policy.

The screenshot shows the 'Access policy' blade for a blob named 'blob3.txt'. Under the 'Immutable blob versions' section, there is a table with one row. The row contains the identifier 'Time-based retention', the retention date '6/26/2021, 4:37:32 PM', and a 'State' column. A context menu is open over the 'Time-based retention' row, listing three options: 'Lock policy' (selected), 'Edit' (with a cursor icon pointing at it), and 'Delete'.

## Configure or clear a legal hold

A legal hold stores immutable data until the legal hold is explicitly cleared. To learn more about legal hold policies, see [Legal holds for immutable blob data](#).

To configure a legal hold on a blob version, you must first enable version-level immutability support on the storage account or container. For more information, see [Enable support for version-level immutability](#).

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To configure a legal hold on a blob version with the Azure portal, follow these steps:

1. Locate the target version, which may be the current version or a previous version of a blob. Select the **More** button and choose **Access policy**.
2. Under the **Immutable blob versions** section, select **Add policy**.
3. Choose **Legal hold** as the policy type, and select **OK** to apply it.

The following image shows a current version of a blob with both a time-based retention policy and legal hold configured.

Identifier	Retention until	State	...
Legal hold	Indefinite	Enabled	...
Time-based retention	7/2/2021, 2:15:53 PM	Unlocked	...

+ Add policy

To clear a legal hold, navigate to the **Access policy** dialog, select the **More** button, and choose **Delete**.

## Next steps

- [Store business-critical blob data with immutable storage](#)
- [Time-based retention policies for immutable blob data](#)
- [Legal holds for immutable blob data](#)

# Configure immutability policies for containers

8/22/2022 • 6 minutes to read • [Edit Online](#)

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 cannot 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. Immutability policies include time-based retention policies and legal holds. For more information about immutability policies for Blob Storage, see [Store business-critical blob data with immutable storage](#).

An immutability policy may be scoped either to an individual blob version or to a container. This article describes how to configure a container-level immutability policy. To learn how to configure version-level immutability policies, see [Configure immutability policies for blob versions](#).

## Configure a retention policy on a container

To configure a time-based retention policy on a container, use the Azure portal, PowerShell, or Azure CLI. You can configure a container-level retention policy for between 1 and 146000 days.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To configure a time-based retention policy on a container with the Azure portal, follow these steps:

1. Navigate to the desired container.
2. Select the **More** button on the right, then select **Access policy**.
3. In the **Immutable blob storage** section, select **Add policy**.
4. In the **Policy type** field, select **Time-based retention**, and specify the retention period in days.
5. To create a policy with container scope, do not check the box for **Enable version-level immutability**.
6. If desired, select **Allow additional protected appends** to enable writes to append blobs that are protected by an immutability policy. For more information, see [Allow protected append blobs writes](#).

## Access policy

sample-container-clw-tbr



Save

X

### Immutable blob storage

Policy type (i)

Set retention period for \* (i)

days

Enable version-level immutability (i)

Allow additional protected appends (i)

OK

Cancel

After you've configured the immutability policy, you will see that it is scoped to the container:

## Access policy

sample-container-clw-tbr

X



Stored access policies

Identifier	Start time	Expiry time	Permissions
No results			

Add policy

Immutable blob storage (i)

Identifier	Scope	Retention interval	State
Time-based retention	Container	14 days	Unlocked

Add policy

## Modify an unlocked retention policy

You can modify an unlocked time-based retention policy to shorten or lengthen the retention interval and to allow additional writes to append blobs in the container. You can also delete an unlocked policy.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To modify an unlocked time-based retention policy in the Azure portal, follow these steps:

1. Navigate to the desired container.
2. Select the **More** button and choose **Access policy**.

3. Under the **Immutable blob versions** section, locate the existing unlocked policy. Select the **More** button, then select **Edit** from the menu.
4. Provide a new retention interval for the policy. You can also select **Allow additional protected appends** to permit writes to protected append blobs.

## Immutable blob storage

- i** Enter the time interval in days that the data needs to be kept in an non-erasable and non-modifiable state. Upon the expiration of the retention interval, the data will continue to be in a non-modifiable state, but can be deleted. Retention policy changes may require some time to take effect.  
[Learn more about immutable blob storage](#)

Update retention period to \*

14

days

Enable version-level immutability i

Allow additional protected appends i

**OK**

**Cancel**

To delete an unlocked policy, select the **More** button, then **Delete**.

### NOTE

You can enable version-level immutability policies by selecting the Enable version-level immutability checkbox. For more information about enabling version-level immutability policies, see [Configure immutability policies for blob versions](#).

## Lock a time-based retention policy

When you have finished testing a time-based retention policy, you can lock the policy. A locked policy is compliant with SEC 17a-4(f) and other regulatory compliance. You can lengthen the retention interval for a locked policy up to five times, but you cannot shorten it.

After a policy is locked, you cannot delete it. However, you can delete the blob after the retention interval has expired.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To lock a policy with the Azure portal, follow these steps:

1. Navigate to a container with an unlocked policy.
2. Under the **Immutable blob versions** section, locate the existing unlocked policy. Select the **More** button, then select **Lock policy** from the menu.
3. Confirm that you want to lock the policy.

## Access policy

sample-container-clw-tbr



Stored access policies

Identifier	Start time	Expiry time	Permissions
No results			



Immutable blob storage ⓘ

Identifier	Scope	Retention interval	State
Time-based retention	Container		Unlocked

+ Add policy

Edit  
 Policy audit  
 Lock policy (Mouse over)  
 Delete

## Configure or clear a legal hold

A legal hold stores immutable data until the legal hold is explicitly cleared. To learn more about legal hold policies, see [Legal holds for immutable blob data](#).

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To configure a legal hold on a container with the Azure portal, follow these steps:

1. Navigate to the desired container.
2. Select the **More** button and choose **Access policy**.
3. Under the **Immutable blob versions** section, select **Add policy**.
4. Choose **Legal hold** as the policy type, and select **OK** to apply it.

The following image shows a container with both a time-based retention policy and legal hold configured.

# Access policy

X

sample-container-policies



Stored access policies

Identifier	Start time	Expiry time	Permissions
No results			

[+ Add policy](#)

Immutable blob storage ⓘ

Identifier	Scope	Retention interval	State
Time-based retention	Container	14 days	Unlocked
Legal hold	Container	Indefinite	Enabled

To clear a legal hold, navigate to the **Access policy** dialog, select the **More** button, and choose **Delete**.

## Next steps

- [Store business-critical blob data with immutable storage](#)
- [Time-based retention policies for immutable blob data](#)
- [Legal holds for immutable blob data](#)
- [Configure immutability policies for blob versions](#)

# Change how a storage account is replicated

8/22/2022 • 11 minutes to read • [Edit Online](#)

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 the [Service-Level Agreement \(SLA\) for Azure Storage](#) even in the face of failures.

Azure Storage offers the following types of replication:

- Locally redundant storage (LRS)
- Zone-redundant storage (ZRS)
- Geo-redundant storage (GRS) or read-access geo-redundant storage (RA-GRS)
- Geo-zone-redundant storage (GZRS) or read-access geo-zone-redundant storage (RA-GZRS)

For an overview of each of these options, see [Azure Storage redundancy](#).

## Switch between types of replication

You can switch a storage account from one type of replication to any other type, but some scenarios are more straightforward than others. If you want to add or remove geo-replication or read access to the secondary region, you can use the Azure portal, PowerShell, or Azure CLI to update the replication setting in some scenarios; other scenarios require a manual or live migration. If you want to change how data is replicated in the primary region, by moving from LRS to ZRS or vice versa, then you must either perform a manual migration or request a live migration. And if you want to move from ZRS to GZRS or RA-GZRS, then you must perform a live migration, unless you are performing a failback operation after failover.

The following table provides an overview of how to switch from each type of replication to another:

SWITCHING	...TO LRS	...TO GRS/RA-GRS	...TO ZRS	...TO GZRS/RA-GZRS
...from LRS	N/A	Use Azure portal, PowerShell, or CLI to change the replication setting <sup>1,2</sup>	Perform a manual migration OR Request a live migration <sup>5</sup>	Perform a manual migration OR Switch to GRS/RA-GRS first and then request a live migration <sup>3</sup>
...from GRS/RA-GRS	Use Azure portal, PowerShell, or CLI to change the replication setting	N/A	Perform a manual migration OR Switch to LRS first and then request a live migration <sup>3</sup>	Perform a manual migration OR Request a live migration <sup>3</sup>

SWITCHING	...TO LRS	...TO GRS/RA-GRS	...TO ZRS	...TO GZRS/RA-GZRS
...from ZRS	Perform a manual migration	Perform a manual migration	N/A	Request a live migration <sup>3</sup>  OR  Use Azure Portal, PowerShell or Azure CLI to change the replication setting as part of a failback operation only <sup>4</sup>
...from GZRS/RA-GZRS	Perform a manual migration	Perform a manual migration	Use Azure portal, PowerShell, or CLI to change the replication setting	N/A

<sup>1</sup> Incurs a one-time egress charge.

<sup>2</sup> Migrating from LRS to GRS is not supported if the storage account contains blobs in the archive tier.

<sup>3</sup> Live migration is supported for standard general-purpose v2 and premium file share storage accounts. Live migration is not supported for premium block blob or page blob storage accounts.

<sup>4</sup> After an account failover to the secondary region, it's possible to initiate a fail back from the new primary back to the new secondary with PowerShell or Azure CLI (version 2.30.0 or later). For more information, see [Use caution when failing back to the original primary](#).

<sup>5</sup> Migrating from LRS to ZRS is not supported if the NFSv3 protocol support is enabled for Azure Blob Storage or if the storage account contains Azure Files NFSv4.1 shares.

#### Caution

If you performed an [account failover](#) for your (RA-)GRS or (RA-)GZRS account, the account is locally redundant (LRS) in the new primary region after the failover. Live migration to ZRS or GZRS for an LRS account resulting from a failover is not supported. This is true even in the case of so-called failback operations. For example, if you perform an account failover from RA-GZRS to the LRS in the secondary region, and then configure it again to RA-GRS and perform another account failover to the original primary region, you can't contact support for the original live migration to RA-GZRS in the primary region. Instead, you'll need to perform a manual migration to ZRS or GZRS.

To change the redundancy configuration for a storage account that contains blobs in the Archive tier, you must first rehydrate all archived blobs to the Hot or Cool tier. Microsoft recommends that you avoid changing the redundancy configuration for a storage account that contains archived blobs if at all possible, because rehydration operations can be costly and time-consuming.

## Change the replication setting

You can use the Azure portal, PowerShell, or Azure CLI to change the replication setting for a storage account, as long as you are not changing how data is replicated in the primary region. If you are migrating from LRS in the primary region to ZRS in the primary region or vice versa, then you must perform either a manual migration or a live migration.

Changing how your storage account is replicated does not result in down time for your applications.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To change the redundancy option for your storage account in the Azure portal, follow these steps:

1. Navigate to your storage account in the Azure portal.

2. Under **Settings** select **Configuration**.

3. Update the **Replication** setting.

The screenshot shows the Azure Storage Configuration page for a storage account named 'exampleaccountnametest'. The 'Replication' section is highlighted with a red box. It displays the current replication setting as 'Locally-redundant storage (LRS)'. Below this, there are two options: 'Locally-redundant storage (LRS)' and 'Geo-redundant storage (GRS)'. A third option, 'Read-access geo-redundant storage (RA-GRS)', is also listed but is not selected. The 'Locally-redundant storage (LRS)' option is selected, indicated by a checked radio button.

## Perform a manual migration to ZRS, GZRS, or RA-GZRS

If you want to change how data in your storage account is replicated in the primary region, by moving from LRS to ZRS or vice versa, then you may opt to perform a manual migration. A manual migration provides more flexibility than a live migration. You control the timing of a manual migration, so use this option if you need the migration to complete by a certain date.

When you perform a manual migration from LRS to ZRS in the primary region or vice versa, the destination storage account can be geo-redundant and can also be configured for read access to the secondary region. For example, you can migrate an LRS account to a GZRS or RA-GZRS account in one step.

You cannot use a manual migration to migrate from ZRS to GZRS or RA-GZRS. You must request a live migration.

A manual migration can result in application downtime. If your application requires high availability, Microsoft also provides a live migration option. A live migration is an in-place migration with no downtime.

With a manual migration, you copy the data from your existing storage account to a new storage account that uses ZRS in the primary region. To perform a manual migration, you can use one of the following options:

- Copy data by using an existing tool such as AzCopy, one of the Azure Storage client libraries, or a reliable third-party tool.
- If you're familiar with Hadoop or HDInsight, you can attach both the source storage account and destination storage account account to your cluster. Then, parallelize the data copy process with a tool like DistCp.

## Request a live migration to ZRS, GZRS, or RA-GZRS

If you need to migrate your storage account from LRS to ZRS in the primary region with no application downtime, you can request a live migration from Microsoft. To migrate from LRS to GZRS or RA-GZRS, first

switch to GRS or RA-GRS and then request a live migration. Similarly, you can request a live migration from ZRS, GRS, or RA-GRS to GZRS or RA-GZRS. To migrate from GRS or RA-GRS to ZRS, first switch to LRS, then request a live migration.

During a live migration, you can access data in your storage account with no loss of durability or availability. The Azure Storage SLA is maintained during the migration process. There is no data loss associated with a live migration. Service endpoints, access keys, shared access signatures, and other account options remain unchanged after the migration.

For standard performance, ZRS supports general-purpose v2 accounts only, so make sure to upgrade your storage account if it is a general-purpose v1 account prior to submitting a request for a live migration to ZRS. For more information, see [Upgrade to a general-purpose v2 storage account](#). A storage account must contain data to be migrated via live migration.

For premium performance, live migration is supported for premium file share accounts, but not for premium block blob or premium page blob accounts.

If your account uses RA-GRS, then you need to first change your account's replication type to either LRS or GRS before proceeding with a live migration. This intermediary step removes the secondary read-only endpoint provided by RA-GRS.

While Microsoft handles your request for live migration promptly, there's no guarantee as to when a live migration will complete. If you need your data migrated to ZRS by a certain date, then Microsoft recommends that you perform a manual migration instead. Generally, the more data you have in your account, the longer it takes to migrate that data.

You must perform a manual migration if:

- You want to migrate your data into a ZRS storage account that is located in a region different than the source account.
- Your storage account is a premium page blob or block blob account.
- You want to migrate data from ZRS to LRS, GRS or RA-GRS.
- Your storage account includes data in the archive tier.

You can request live migration through the [Azure Support portal](#).

#### IMPORTANT

If you need to migrate more than one storage account, create a single support ticket and specify the names of the accounts to convert on the **Details** tab.

Follow these steps to request a live migration:

1. In the Azure portal, navigate to a storage account that you want to migrate.
2. Under **Support + troubleshooting**, select **New Support Request**.
3. Complete the **Basics** tab based on your account information:
  - **Issue type:** Select **Technical**.
  - **Service:** Select **My Services**, then **Storage Account Management**.
  - **Resource:** Select a storage account to migrate. If you need to specify multiple storage accounts, you can do so in the **Details** section.
  - **Problem type:** Choose **Data Migration**.
  - **Problem subtype:** Choose **Migrate to ZRS, GZRS, or RA-GZRS**.

Dashboard > storagesamples

## storagesamples | New support request

Storage account

Search (Ctrl+ /)

Diagnostic settings (preview)

Logs (preview)

Advisor recommendations

Monitoring (classic)

Alerts (classic)

Metrics (classic)

Diagnostic settings (classic)

Usage (classic)

Automation

Tasks (preview)

Export template

Support + troubleshooting

Resource health

Connectivity check

Recover deleted account

New support request

Basics Solutions Details Review + create

Create a new support request to get assistance with billing, subscription, technical (including advisory) or quota management issues.

Complete the Basics tab by selecting the options that best describe your problem. Providing detailed, accurate information can help to solve your issues faster.

Summary \*

Request live migration

Issue type \*

Technical

Subscription \*

<Subscription>

Can't find your subscription? Show more ⓘ

Service

My services (radio button selected)

All services (radio button)

Service type \*

Storage Account Management

Resource \*

storagesamples

Problem type \*

Data Migration

Problem subtype \*

Migrate to ZRS, GZRS or RA-GZRS

Next: Solutions >>

4. Select **Next**. On the **Solutions** tab, you can check the eligibility of your storage accounts for migration.
5. Select **Next**. If you have more than one storage account to migrate, then on the **Details** tab, specify the name for each account, separated by a semicolon.

Dashboard > storagesamples

## storagesamples | New support request

Storage account

Search (Ctrl+ /)

Alerts

Metrics

Workbooks

Diagnostic settings (preview)

Logs (preview)

Advisor recommendations

Monitoring (classic)

Alerts (classic)

Metrics (classic)

Diagnostic settings (classic)

Usage (classic)

Automation

Tasks (preview)

Export template

Support + troubleshooting

Resource health

Connectivity check

Recover deleted account

New support request

Basics Solutions Details Review + create

Information provided on this tab will be used to further assess your issue and help the support engineer troubleshoot the problem. Verify the contact information before moving to the Review + Create.

**Problem details**

Target replication type \* To RA-GZRS

Storage accounts from \* storagesamples;storagesecuritysamples;storagesecuritypolicies

Approximate start time of the most recent occurrence MM/DD/YYYY Enter in local time

Provide any additional details \*

Details...

File upload Select a file

<< Previous: Solutions Next: Review + create >>

6. Fill out the additional required information on the **Details** tab, then select **Review + create** to review and submit your support ticket. A support person will contact you to provide any assistance you may need.

#### NOTE

Premium file shares are available only for LRS and ZRS.

GZRS storage accounts do not currently support the archive tier. See [Hot, Cool, and Archive access tiers for blob data](#) for more details.

Managed disks are only available for LRS and cannot be migrated to ZRS. You can store snapshots and images for standard SSD managed disks on standard HDD storage and [choose between LRS and ZRS options](#). For information about integration with availability sets, see [Introduction to Azure managed disks](#).

## Switch from ZRS Classic

#### IMPORTANT

Microsoft will deprecate and migrate ZRS Classic accounts on March 31, 2021. More details will be provided to ZRS Classic customers before deprecation.

After ZRS becomes generally available in a given region, customers will no longer be able to create ZRS Classic accounts from the Azure portal in that region. Using Microsoft PowerShell and Azure CLI to create ZRS Classic accounts is an option until ZRS Classic is deprecated. For information about where ZRS is available, see [Azure Storage redundancy](#).

ZRS Classic asynchronously replicates data across data centers within one to two regions. Replicated data may not be available unless Microsoft initiates failover to the secondary. A ZRS Classic account can't be converted to or from LRS, GRS, or RA-GRS. ZRS Classic accounts also don't support metrics or logging.

ZRS Classic is available only for **block blobs** in general-purpose V1 (GPv1) storage accounts. For more information about storage accounts, see [Azure storage account overview](#).

To manually migrate ZRS account data to or from an LRS, GRS, RA-GRS, or ZRS Classic account, use one of the following tools: AzCopy, Azure Storage Explorer, PowerShell, or Azure CLI. You can also build your own migration solution with one of the Azure Storage client libraries.

You can also upgrade your ZRS Classic storage account to ZRS by using the Azure portal, PowerShell, or Azure CLI in regions where ZRS is available.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To upgrade to ZRS in the Azure portal, navigate to the **Configuration** settings of the account and choose **Upgrade**:

The cost of your storage account depends on the usage and the options you choose below.

[Learn more](#)

Account kind

Storage (general purpose v1)



This account can be upgraded to a General Purpose v2 account with additional features. Upgrading is permanent and will result in billing changes. [Learn more](#).



[Upgrade](#)

Performance [i](#)

[Standard](#) [Premium](#)

\* Secure transfer required [i](#)

[Disabled](#) [Enabled](#)

Replication [i](#)

[Zone-redundant storage \(ZRS classic\)](#)



The replication setting for a storage accounts using ZRS can't be changed.

Data Lake Storage Gen2

Hierarchical namespace [i](#)

[Disabled](#) [Enabled](#)

## Costs associated with changing how data is replicated

The costs associated with changing how data is replicated depend on your conversion path. Ordering from least to the most expensive, Azure Storage redundancy offerings include LRS, ZRS, GRS, RA-GRS, GZRS, and RA-GZRS.

For example, going *from* LRS to any other type of replication will incur additional charges because you are moving to a more sophisticated redundancy level. Migrating *to* GRS or RA-GRS will incur an egress bandwidth charge at the time of migration because your entire storage account is being replicated to the secondary region. All subsequent writes to the primary region also incur egress bandwidth charges to replicate the write to the secondary region. For details on bandwidth charges, see [Azure Storage Pricing page](#).

If you migrate your storage account from GRS to LRS, there is no additional cost, but your replicated data is deleted from the secondary location.

### IMPORTANT

If you migrate your storage account from RA-GRS to GRS or LRS, that account is billed as RA-GRS for an additional 30 days beyond the date that it was converted.

## See also

- [Azure Storage redundancy](#)
- [Check the Last Sync Time property for a storage account](#)
- [Use geo-redundancy to design highly available applications](#)

# Use geo-redundancy to design highly available applications

8/22/2022 • 19 minutes to read • [Edit Online](#)

A common feature of cloud-based infrastructures like Azure Storage is that they provide a highly available and durable platform for hosting data and applications. Developers of cloud-based applications must consider carefully how to leverage this platform to maximize those advantages for their users. Azure Storage offers geo-redundant storage to ensure high availability even in the event of a regional outage. Storage accounts configured for geo-redundant replication are synchronously replicated in the primary region, and then asynchronously replicated to a secondary region that is hundreds of miles away.

Azure Storage offers two options for geo-redundant replication. The only difference between these two options is how data is replicated in the primary region:

- **Geo-zone-redundant storage (GZRS):** Data is replicated synchronously across three Azure availability zones in the primary region using *zone-redundant storage (ZRS)*, then replicated asynchronously to the secondary region. For read access to data in the secondary region, enable read-access geo-zone-redundant storage (RA-GZRS).

Microsoft recommends using GZRS/RA-GZRS for scenarios that require maximum availability and durability.

- **Geo-redundant storage (GRS):** Data is replicated synchronously three times in the primary region using *locally redundant storage (LRS)*, then replicated asynchronously to the secondary region. For read access to data in the secondary region, enable read-access geo-redundant storage (RA-GRS).

This article shows how to design your application to handle an outage in the primary region. If the primary region becomes unavailable, your application can adapt to perform read operations against the secondary region instead. Make sure that your storage account is configured for RA-GRS or RA-GZRS before you get started.

## Application design considerations when reading from the secondary

The purpose of this article is to show you how to design an application that will continue to function (albeit in a limited capacity) even in the event of a major disaster at the primary data center. You can design your application to handle transient or long-running issues by reading from the secondary region when there is a problem that interferes with reading from the primary region. When the primary region is available again, your application can return to reading from the primary region.

Keep in mind these key points when designing your application for RA-GRS or RA-GZRS:

- Azure Storage maintains a read-only copy of the data you store in your primary region in a secondary region. As noted above, the storage service determines the location of the secondary region.
- The read-only copy is **eventually consistent** with the data in the primary region.
- For blobs, tables, and queues, you can query the secondary region for a *Last Sync Time* value that tells you when the last replication from the primary to the secondary region occurred. (This is not supported for Azure Files, which doesn't have RA-GRS redundancy at this time.)
- You can use the Storage Client Library to read and write data in either the primary or secondary region. You can also redirect read requests automatically to the secondary region if a read request to the primary

region times out.

- If the primary region becomes unavailable, you can initiate an account failover. When you fail over to the secondary region, the DNS entries pointing to the primary region are changed to point to the secondary region. After the failover is complete, write access is restored for GRS and RA-GRS accounts. For more information, see [Disaster recovery and storage account failover](#).

### Using eventually consistent data

The proposed solution assumes that it is acceptable to return potentially stale data to the calling application. Because data in the secondary region is eventually consistent, it is possible the primary region may become inaccessible before an update to the secondary region has finished replicating.

For example, suppose your customer submits an update successfully, but the primary region fails before the update is propagated to the secondary region. When the customer asks to read the data back, they receive the stale data from the secondary region instead of the updated data. When designing your application, you must decide whether this is acceptable, and if so, how you will message the customer.

Later in this article, we show how to check the Last Sync Time for the secondary data to check whether the secondary is up-to-date.

### Handling services separately or all together

While unlikely, it is possible for one service to become unavailable while the other services are still fully functional. You can handle the retries and read-only mode for each service separately (blobs, queues, tables), or you can handle retries generically for all the storage services together.

For example, if you use queues and blobs in your application, you may decide to put in separate code to handle retryable errors for each of these. Then if you get a retry from the blob service, but the queue service is still working, only the part of your application that handles blobs will be impacted. If you decide to handle all storage service retries generically and a call to the blob service returns a retryable error, then requests to both the blob service and the queue service will be impacted.

Ultimately, this depends on the complexity of your application. You may decide not to handle the failures by service, but instead to redirect read requests for all storage services to the secondary region and run the application in read-only mode when you detect a problem with any storage service in the primary region.

### Other considerations

These are the other considerations we will discuss in the rest of this article.

- Handling retries of read requests using the Circuit Breaker pattern
- Eventually-consistent data and the Last Sync Time
- Testing

## Running your application in read-only mode

To effectively prepare for an outage in the primary region, you must be able to handle both failed read requests and failed update requests (with update in this case meaning inserts, updates, and deletions). If the primary region fails, read requests can be redirected to the secondary region. However, update requests cannot be redirected to the secondary because the secondary is read-only. For this reason, you need to design your application to run in read-only mode.

For example, you can set a flag that is checked before any update requests are submitted to Azure Storage. When one of the update requests comes through, you can skip it and return an appropriate response to the customer. You may even want to disable certain features altogether until the problem is resolved and notify users that those features are temporarily unavailable.

If you decide to handle errors for each service separately, you will also need to handle the ability to run your application in read-only mode by service. For example, you may have read-only flags for each service that can be enabled and disabled. Then you can handle the flag in the appropriate places in your code.

Being able to run your application in read-only mode has another side benefit – it gives you the ability to ensure limited functionality during a major application upgrade. You can trigger your application to run in read-only mode and point to the secondary data center, ensuring nobody is accessing the data in the primary region while you're making upgrades.

## Handling updates when running in read-only mode

There are many ways to handle update requests when running in read-only mode. We won't cover this comprehensively, but generally, there are a couple of patterns that you consider.

- You can respond to your user and tell them you are not currently accepting updates. For example, a contact management system could enable customers to access contact information but not make updates.
- You can enqueue your updates in another region. In this case, you would write your pending update requests to a queue in a different region, and then have a way to process those requests after the primary data center comes online again. In this scenario, you should let the customer know that the update requested is queued for later processing.
- You can write your updates to a storage account in another region. Then when the primary data center comes back online, you can have a way to merge those updates into the primary data, depending on the structure of the data. For example, if you are creating separate files with a date/time stamp in the name, you can copy those files back to the primary region. This works for some workloads such as logging and IoT data.

## Handling retries

The Azure Storage client library helps you determine which errors can be retried. For example, a 404 error (resource not found) would not be retried because retrying it is not likely to result in success. On the other hand, a 500 error can be retried because it is a server error, and the problem may simply be a transient issue. For more details, check out the [open source code for the ExponentialRetry class](#) in the .NET storage client library. (Look for the ShouldRetry method.)

### Read requests

Read requests can be redirected to secondary storage if there is a problem with primary storage. As noted above in [Using Eventually Consistent Data](#), it must be acceptable for your application to potentially read stale data. If you are using the storage client library to access data from the secondary, you can specify the retry behavior of a read request by setting a value for the **LocationMode** property to one of the following:

- **PrimaryOnly** (the default)
- **PrimaryThenSecondary**
- **SecondaryOnly**
- **SecondaryThenPrimary**

When you set the **LocationMode** to **PrimaryThenSecondary**, if the initial read request to the primary endpoint fails with an error that can be retried, the client automatically makes another read request to the secondary endpoint. If the error is a server timeout, then the client will have to wait for the timeout to expire before it receives a retryable error from the service.

There are basically two scenarios to consider when you are deciding how to respond to a retryable error:

- This is an isolated problem and subsequent requests to the primary endpoint will not return a retryable error. An example of where this might happen is when there is a transient network error.

In this scenario, there is no significant performance penalty in having **LocationMode** set to **PrimaryThenSecondary** as this only happens infrequently.

- This is a problem with at least one of the storage services in the primary region and all subsequent requests to that service in the primary region are likely to return retryable errors for a period of time. An example of this is if the primary region is completely inaccessible.

In this scenario, there is a performance penalty because all your read requests will try the primary endpoint first, wait for the timeout to expire, then switch to the secondary endpoint.

For these scenarios, you should identify that there is an ongoing issue with the primary endpoint and send all read requests directly to the secondary endpoint by setting the **LocationMode** property to **SecondaryOnly**. At this time, you should also change the application to run in read-only mode. This approach is known as the [Circuit Breaker Pattern](#).

### Update requests

The Circuit Breaker pattern can also be applied to update requests. However, update requests cannot be redirected to secondary storage, which is read-only. For these requests, you should leave the **LocationMode** property set to **PrimaryOnly** (the default). To handle these errors, you can apply a metric to these requests – such as 10 failures in a row – and when your threshold is met, switch the application into read-only mode. You can use the same methods for returning to update mode as those described below in the next section about the Circuit Breaker pattern.

## Circuit Breaker pattern

Using the Circuit Breaker pattern in your application can prevent it from retrying an operation that is likely to fail repeatedly. It allows the application to continue to run rather than taking up time while the operation is retried exponentially. It also detects when the fault has been fixed, at which time the application can try the operation again.

### How to implement the circuit breaker pattern

To identify that there is an ongoing problem with a primary endpoint, you can monitor how frequently the client encounters retryable errors. Because each case is different, you have to decide on the threshold you want to use for the decision to switch to the secondary endpoint and run the application in read-only mode. For example, you could decide to perform the switch if there are 10 failures in a row with no successes. Another example is to switch if 90% of the requests in a 2-minute period fail.

For the first scenario, you can simply keep a count of the failures, and if there is a success before reaching the maximum, set the count back to zero. For the second scenario, one way to implement it is to use the **MemoryCache** object (in .NET). For each request, add a **Cacheltem** to the cache, set the value to success (1) or fail (0), and set the expiration time to 2 minutes from now (or whatever your time constraint is). When an entry's expiration time is reached, the entry is automatically removed. This will give you a rolling 2-minute window. Each time you make a request to the storage service, you first use a Linq query across the **MemoryCache** object to calculate the percent success by summing the values and dividing by the count. When the percent success drops below some threshold (such as 10%), set the **LocationMode** property for read requests to **SecondaryOnly** and switch the application into read-only mode before continuing.

The threshold of errors used to determine when to make the switch may vary from service to service in your application, so you should consider making them configurable parameters. This is also where you decide to handle retryable errors from each service separately or as one, as discussed previously.

Another consideration is how to handle multiple instances of an application, and what to do when you detect retryable errors in each instance. For example, you may have 20 VMs running with the same application loaded.

Do you handle each instance separately? If one instance starts having problems, do you want to limit the response to just that one instance, or do you want to try to have all instances respond in the same way when one instance has a problem? Handling the instances separately is much simpler than trying to coordinate the response across them, but how you do this depends on your application's architecture.

### Options for monitoring the error frequency

You have three main options for monitoring the frequency of retries in the primary region in order to determine when to switch over to the secondary region and change the application to run in read-only mode.

- Add a handler for the [Retrying](#) event on the [OperationContext](#) object you pass to your storage requests – this is the method displayed in this article and used in the accompanying sample. These events fire whenever the client retries a request, enabling you to track how often the client encounters retryable errors on a primary endpoint.

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

We are currently working to create code snippets reflecting version 12.x of the Azure Storage client libraries. For more information, see [Announcing the Azure Storage v12 Client Libraries](#).

- In the [Evaluate](#) method in a custom retry policy, you can run custom code whenever a retry takes place. In addition to recording when a retry happens, this also gives you the opportunity to modify your retry behavior.

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

We are currently working to create code snippets reflecting version 12.x of the Azure Storage client libraries. For more information, see [Announcing the Azure Storage v12 Client Libraries](#).

- The third approach is to implement a custom monitoring component in your application that continually pings your primary storage endpoint with dummy read requests (such as reading a small blob) to determine its health. This would take up some resources, but not a significant amount. When a problem is discovered that reaches your threshold, you would then perform the switch to [SecondaryOnly](#) and read-only mode.

At some point, you will want to switch back to using the primary endpoint and allowing updates. If using one of the first two methods listed above, you could simply switch back to the primary endpoint and enable update mode after an arbitrarily selected amount of time or number of operations has been performed. You can then let it go through the retry logic again. If the problem has been fixed, it will continue to use the primary endpoint and allow updates. If there is still a problem, it will once more switch back to the secondary endpoint and read-only mode after failing the criteria you've set.

For the third scenario, when pinging the primary storage endpoint becomes successful again, you can trigger the switch back to [PrimaryOnly](#) and continue allowing updates.

## Handling eventually consistent data

Geo-redundant storage works by replicating transactions from the primary to the secondary region. This replication process guarantees that the data in the secondary region is *eventually consistent*. This means that all the transactions in the primary region will eventually appear in the secondary region, but that there may be a lag before they appear, and that there is no guarantee the transactions arrive in the secondary region in the same order as that in which they were originally applied in the primary region. If your transactions arrive in the secondary region out of order, you *may* consider your data in the secondary region to be in an inconsistent state until the service catches up.

The following table shows an example of what might happen when you update the details of an employee to make them a member of the *administrators* role. For the sake of this example, this requires you update the **employee** entity and update an **administrator role** entity with a count of the total number of administrators. Notice how the updates are applied out of order in the secondary region.

TIME	TRANSACTION	REPLICATION	LAST SYNC TIME	RESULT
T0	Transaction A: Insert employee entity in primary			Transaction A inserted to primary, not replicated yet.
T1		Transaction A replicated to secondary	T1	Transaction A replicated to secondary. Last Sync Time updated.
T2	Transaction B: Update employee entity in primary		T1	Transaction B written to primary, not replicated yet.
T3	Transaction C: Update administrator role entity in primary		T1	Transaction C written to primary, not replicated yet.
T4		Transaction C replicated to secondary	T1	Transaction C replicated to secondary. LastSyncTime not updated because transaction B has not been replicated yet.
T5	Read entities from secondary		T1	You get the stale value for employee entity because transaction B hasn't replicated yet. You get the new value for administrator role entity because C has replicated. Last Sync Time still hasn't been updated because transaction B hasn't replicated. You can tell the administrator role entity is inconsistent because the entity date/time is after the Last Sync Time.

TIME	TRANSACTION	REPLICATION	LAST SYNC TIME	RESULT
T6		Transaction B replicated to secondary	T6	T6 – All transactions through C have been replicated, Last Sync Time is updated.

In this example, assume the client switches to reading from the secondary region at T5. It can successfully read the **administrator role** entity at this time, but the entity contains a value for the count of administrators that is not consistent with the number of **employee** entities that are marked as administrators in the secondary region at this time. Your client could simply display this value, with the risk that it is inconsistent information.

Alternatively, the client could attempt to determine that the **administrator role** is in a potentially inconsistent state because the updates have happened out of order, and then inform the user of this fact.

To recognize that it has potentially inconsistent data, the client can use the value of the *Last Sync Time* that you can get at any time by querying a storage service. This tells you the time when the data in the secondary region was last consistent and when the service had applied all the transactions prior to that point in time. In the example shown above, after the service inserts the **employee** entity in the secondary region, the last sync time is set to *T1*. It remains at *T1* until the service updates the **employee** entity in the secondary region when it is set to *T6*. If the client retrieves the last sync time when it reads the entity at *T5*, it can compare it with the timestamp on the entity. If the timestamp on the entity is later than the last sync time, then the entity is in a potentially inconsistent state, and you can take whatever is the appropriate action for your application. Using this field requires that you know when the last update to the primary was completed.

To learn how to check the last sync time, see [Check the Last Sync Time property for a storage account](#).

## Testing

It's important to test that your application behaves as expected when it encounters retryable errors. For example, you need to test that the application switches to the secondary and into read-only mode when it detects a problem, and switches back when the primary region becomes available again. To do this, you need a way to simulate retryable errors and control how often they occur.

You can use [Fiddler](#) to intercept and modify HTTP responses in a script. This script can identify responses that come from your primary endpoint and change the HTTP status code to one that the Storage Client Library recognizes as a retryable error. This code snippet shows a simple example of a Fiddler script that intercepts responses to read requests against the **employeedata** table to return a 502 status:

- [Java v12 SDK](#)
- [Java v11 SDK](#)

We are currently working to create code snippets reflecting version 12.x of the Azure Storage client libraries. For more information, see [Announcing the Azure Storage v12 Client Libraries](#).

## Next steps

For a complete sample showing how to make the switch back and forth between the primary and secondary endpoints, see [Azure Samples – Using the Circuit Breaker Pattern with RA-GRS storage](#).

# Check the Last Sync Time property for a storage account

8/22/2022 • 2 minutes to read • [Edit Online](#)

When you configure a storage account, you can specify that your data is copied to a secondary region that is hundreds of miles from the primary region. Geo-replication offers durability for your data in the event of a significant outage in the primary region, such as a natural disaster. If you additionally enable read access to the secondary region, your data remains available for read operations if the primary region becomes unavailable. You can design your application to switch seamlessly to reading from the secondary region if the primary region is unresponsive.

Geo-redundant storage (GRS) and geo-zone-redundant storage (GZRS) both replicate your data asynchronously to a secondary region. For read access to the secondary region, enable read-access geo-redundant storage (RA-GRS) or read-access geo-zone-redundant storage (RA-GZRS). For more information about the various options for redundancy offered by Azure Storage, see [Azure Storage redundancy](#).

This article describes how to check the **Last Sync Time** property for your storage account so that you can evaluate any discrepancy between the primary and secondary regions.

## About the Last Sync Time property

Because geo-replication is asynchronous, it is possible that data written to the primary region has not yet been written to the secondary region at the time an outage occurs. The **Last Sync Time** property indicates the last time that data from the primary region was written successfully to the secondary region. All writes made to the primary region before the last sync time are available to be read from the secondary location. Writes made to the primary region after the last sync time property may or may not be available for reads yet.

The **Last Sync Time** property is a GMT date/time value.

## Get the Last Sync Time property

You can use PowerShell or Azure CLI to retrieve the value of the **Last Sync Time** property.

- [PowerShell](#)
- [Azure CLI](#)

To get the last sync time for the storage account with PowerShell, install version 1.11.0 or later of the [Az.Storage](#) module. Then check the storage account's **GeoReplicationStats.LastSyncTime** property. Remember to replace the placeholder values with your own values:

```
$lastSyncTime = $(Get-AzStorageAccount -ResourceGroupName <resource-group> `
 -Name <storage-account> `
 -IncludeGeoReplicationStats).GeoReplicationStats.LastSyncTime
```

## See also

- [Azure Storage redundancy](#)
- [Change the redundancy option for a storage account](#)
- [Use geo-redundancy to design highly available applications](#)

# Initiate a storage account failover

8/22/2022 • 4 minutes to read • [Edit Online](#)

If the primary endpoint for your geo-redundant storage account becomes unavailable for any reason, you can initiate an account failover. An account failover updates the secondary endpoint to become the primary endpoint for your storage account. Once the failover is complete, clients can begin writing to the new primary region. Forced failover enables you to maintain high availability for your applications.

This article shows how to initiate an account failover for your storage account using the Azure portal, PowerShell, or Azure CLI. To learn more about account failover, see [Disaster recovery and storage account failover](#).

## WARNING

An account failover typically results in some data loss. To understand the implications of an account failover and to prepare for data loss, review [Understand the account failover process](#).

## NOTE

This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

## Prerequisites

Before you can perform an account failover on your storage account, make sure that your storage account is configured for geo-replication. Your storage account can use any of the following redundancy options:

- Geo-redundant storage (GRS) or read-access geo-redundant storage (RA-GRS)
- Geo-zone-redundant storage (GZRS) or read-access geo-zone-redundant storage (RA-GZRS)

For more information about Azure Storage redundancy, see [Azure Storage redundancy](#).

Keep in mind that the following features and services are not supported for account failover:

- Azure File Sync does not support storage account failover. Storage accounts containing Azure file shares being used as cloud endpoints in Azure File Sync should not be failed over. Doing so will cause sync to stop working and may also cause unexpected data loss in the case of newly tiered files.
- Storage accounts that have hierarchical namespace enabled (such as for Data Lake Storage Gen2) are not supported at this time.
- A storage account containing premium block blobs cannot be failed over. Storage accounts that support premium block blobs do not currently support geo-redundancy.
- A storage account containing any [WORM immutability policy](#) enabled containers cannot be failed over. Unlocked/locked time-based retention or legal hold policies prevent failover in order to maintain compliance.

## Initiate the failover

- [Portal](#)
- [PowerShell](#)

- Azure CLI

To initiate an account failover from the Azure portal, follow these steps:

1. Navigate to your storage account.
2. Under **Settings**, select **Geo-replication**. The following image shows the geo-replication and failover status of a storage account.

Azure Storage replication copies your data so that it is protected from transient hardware failures, network or power outages, and natural disasters. If an outage renders the primary region unavailable, then you can initiate a failover to the secondary region to rapidly restore write access to your data. [Learn more](#)

Replication  
Geo-redundant storage (GRS)

Last failover time: -

Storage endpoints: [View all](#)

LOCATION	DATA CENTER TYPE	STATUS	FAILOVER
US West 2	Primary	Available	-
US West Central	Secondary	Available	-

**Prepare for failover**

3. Verify that your storage account is configured for geo-redundant storage (GRS) or read-access geo-redundant storage (RA-GRS). If it's not, then select **Configuration** under **Settings** to update your account to be geo-redundant.
4. The **Last Sync Time** property indicates how far the secondary is behind from the primary. **Last Sync Time** provides an estimate of the extent of data loss that you will experience after the failover is completed. For more information about checking the **Last Sync Time** property, see [Check the Last Sync Time property for a storage account](#).
5. Select **Prepare for failover**.
6. Review the confirmation dialog. When you are ready, enter **Yes** to confirm and initiate the failover.

## Failover

X

The storage account secondary endpoint will become the primary endpoint after failover completes. Please understand the following impact to your storage account before you initiate the failover:

- Your last sync time is **6/11/2020, 6:24:21 PM**. You may lose any data after this time if you initiate the failover.
- After the failover completes, your storage account replication will be converted to a locally-redundant storage (LRS) account. You can convert your account to use geo-redundant storage (GRS) or read-access geo-redundant storage (RA-GRS).
- Once you re-enable GRS or RA-GRS replication on your storage account, Microsoft will replicate data to your new secondary region. Replication time is dependent on the amount of data to be replicated. Please note that there are bandwidth charges for this data transfer. [Learn more](#).

Confirm failover

yes



[Failover](#)

[Cancel](#)

## Important implications of account failover

When you initiate an account failover for your storage account, the DNS records for the secondary endpoint are updated so that the secondary endpoint becomes the primary endpoint. Make sure that you understand the potential impact to your storage account before you initiate a failover.

To estimate the extent of likely data loss before you initiate a failover, check the **Last Sync Time** property. For more information about checking the **Last Sync Time** property, see [Check the Last Sync Time property for a storage account](#).

The time it takes to failover after initiation can vary though typically less than one hour.

After the failover, your storage account type is automatically converted to locally redundant storage (LRS) in the new primary region. You can re-enable geo-redundant storage (GRS) or read-access geo-redundant storage (RA-GRS) for the account. Note that converting from LRS to GRS or RA-GRS incurs an additional cost. The cost is due to the network egress charges to re-replicate the data to the new secondary region. For additional information, see [Bandwidth Pricing Details](#).

After you re-enable GRS for your storage account, Microsoft begins replicating the data in your account to the new secondary region. Replication time depends on many factors, which include:

- The number and size of the objects in the storage account. Many small objects can take longer than fewer and larger objects.
- The available resources for background replication, such as CPU, memory, disk, and WAN capacity. Live traffic takes priority over geo replication.
- If using Blob storage, the number of snapshots per blob.
- If using Table storage, the [data partitioning strategy](#). The replication process can't scale beyond the number of partition keys that you use.

## Next steps

- [Disaster recovery and storage account failover](#)

- Check the Last Sync Time property for a storage account
- Use geo-redundancy to design highly available applications
- Tutorial: Build a highly available application with Blob storage

# Set a blob's access tier

8/22/2022 • 12 minutes to read • [Edit Online](#)

You can set a blob's access tier in any of the following ways:

- By setting the default online access tier (Hot or Cool) for the storage account. Blobs in the account inherit this access tier unless you explicitly override the setting for an individual blob.
- By explicitly setting a blob's tier on upload. You can create a blob in the Hot, Cool, or Archive tier.
- By changing an existing blob's tier with a Set Blob Tier operation, typically to move from a hotter tier to a cooler one.
- By copying a blob with a Copy Blob operation, typically to move from a cooler tier to a hotter one.

This article describes how to manage a blob in an online access tier (Hot or Cool). For more information about how to move a blob to the Archive tier, see [Archive a blob](#). For more information about how to rehydrate a blob from the Archive tier, see [Rehydrate an archived blob to an online tier](#).

For more information about access tiers for blobs, see [Hot, Cool, and Archive access tiers for blob data](#).

## Set the default access tier for a storage account

The default access tier setting for a general-purpose v2 storage account determines in which online tier a new blob is created by default. You can set the default access tier for a general-purpose v2 storage account at the time that you create the account or by updating an existing account's configuration.

When you change the default access tier setting for an existing general-purpose v2 storage account, the change applies to all blobs in the account for which an access tier hasn't been explicitly set. Changing the default access tier may have a billing impact. For details, see [Default account access tier setting](#).

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [AzCopy](#)

To set the default access tier for a storage account at create time in the Azure portal, follow these steps:

1. Navigate to the **Storage accounts** page, and select the **Create** button.
2. Fill out the **Basics** tab.
3. On the **Advanced** tab, under **Blob storage**, set the **Access tier** to either *Hot* or *Cool*. The default setting is *Hot*.
4. Select **Review + Create** to validate your settings and create your storage account.

Access tier 

- Hot:** Frequently accessed data and day-to-day usage scenarios  
 **Cool:** Infrequently accessed data and backup scenarios

To update the default access tier for an existing storage account in the Azure portal, follow these steps:

1. Navigate to the storage account in the Azure portal.
2. Under **Settings**, select **Configuration**.

3. Locate the **Blob access tier (default)** setting, and select either *Hot* or *Cool*. The default setting is *Hot*, if you have not previously set this property.
4. Save your changes.

## Set a blob's tier on upload

When you upload a blob to Azure Storage, you have two options for setting the blob's tier on upload:

- You can explicitly specify the tier in which the blob will be created. This setting overrides the default access tier for the storage account. You can set the tier for a blob or set of blobs on upload to Hot, Cool, or Archive.
- You can upload a blob without specifying a tier. In this case, the blob will be created in the default access tier specified for the storage account (either Hot or Cool).

If you are uploading a new blob that uses an encryption scope, you cannot change the access tier for that blob.

The following sections describe how to specify that a blob is uploaded to either the Hot or Cool tier. For more information about archiving a blob on upload, see [Archive blobs on upload](#).

### Upload a blob to a specific online tier

To create a blob in the Hot or Cool tier, specify that tier when you create the blob. The access tier specified on upload overrides the default access tier for the storage account.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [AzCopy](#)

To upload a blob or set of blobs to a specific tier from the Azure portal, follow these steps:

1. Navigate to the target container.
2. Select the **Upload** button.
3. Select the file or files to upload.
4. Expand the **Advanced** section, and set the **Access tier** to *Hot* or *Cool*.
5. Select the **Upload** button.



### Upload a blob to the default tier

Storage accounts have a default access tier setting that indicates in which online tier a new blob is created. The default access tier setting can be set to either hot or cool. The behavior of this setting is slightly different depending on the type of storage account:

- The default access tier for a new general-purpose v2 storage account is set to the Hot tier by default. You can change the default access tier setting when you create a storage account or after it's created.
- When you create a legacy Blob Storage account, you must specify the default access tier setting as Hot or Cool when you create the storage account. You can change the default access tier setting for the storage account after it's created.

A blob that doesn't have an explicitly assigned tier infers its tier from the default account access tier setting. You can determine whether a blob's access tier is inferred by using the Azure portal, PowerShell, or Azure CLI.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [AzCopy](#)

If a blob's access tier is inferred from the default account access tier setting, then the Azure portal displays the access tier as **Hot (inferred)** or **Cool (inferred)**.

**Authentication method:** Access key ([Switch to Azure AD User Account](#))  
**Location:** container5

Search blobs by prefix (case-sensitive)  Show deleted blobs

Add filter

Name	Modified	Access tier	Blob type	Size	Lease state
blob1.txt	10/25/2021, 2:01:25 ...	Hot (Inferred)	Block blob	48 B	Available
blob2.txt	10/25/2021, 2:01:25 ...	Cool	Block blob	20 B	Available
blob3.txt	10/25/2021, 2:01:25 ...	Cool	Block blob	9 B	Available
blob4.txt	10/25/2021, 2:01:25 ...	Hot (Inferred)	Block blob	9 B	Available
blob5.txt	10/25/2021, 2:01:25 ...	Hot (Inferred)	Block blob	9 B	Available
blob6.txt	10/25/2021, 2:01:26 ...	Hot (Inferred)	Block blob	9 B	Available
blob7.txt	10/25/2021, 2:01:25 ...	Hot (Inferred)	Block blob	9 B	Available
blob8.txt	10/25/2021, 2:01:25 ...	Hot (Inferred)	Block blob	9 B	Available
blob9.txt	10/25/2021, 2:01:25 ...	Hot (Inferred)	Block blob	9 B	Available

## Move a blob to a different online tier

You can move a blob to a different online tier in one of two ways:

- By changing the access tier.
- By copying the blob to a different online tier.

For more information about each of these options, see [Setting or changing a blob's tier](#).

Use PowerShell, Azure CLI, AzCopy v10, or one of the Azure Storage client libraries to move a blob to a different tier.

### Change a blob's tier

When you change a blob's tier, you move that blob and all of its data to the target tier by calling the [Set Blob Tier](#) operation (either directly or via a [lifecycle management](#) policy), or by using the [azcopy set-properties](#) command with AzCopy. This option is typically the best when you're changing a blob's tier from a hotter tier to a cooler one.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [AzCopy](#)

To change a blob's tier from Hot to Cool in the Azure portal, follow these steps:

1. Navigate to the blob for which you want to change the tier.
2. Select the blob, then select the **Change tier** button.
3. In the **Change tier** dialog, select the target tier.
4. Select the **Save** button.

## Change tier

X

blob4.txt

Optimize storage costs by placing your data in the appropriate access tier. [Learn more ↗](#)

Access tier

Cool



**Save**

**Cancel**

### Copy a blob to a different online tier

Call [Copy Blob](#) operation to copy a blob from one tier to another. When you copy a blob to a different tier, you move that blob and all of its data to the target tier. The source blob remains in the original tier, and a new blob is created in the target tier. Calling [Copy Blob](#) is recommended for most scenarios where you're moving a blob from Cool to Hot, or rehydrating a blob from the Archive tier.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [AzCopy](#)

N/A

### Next steps

- [Hot, Cool, and Archive access tiers for blob data](#)
- [Archive a blob](#)
- [Rehydrate an archived blob to an online tier](#)

# Archive a blob

8/22/2022 • 12 minutes to read • [Edit Online](#)

The Archive tier is an offline tier for storing blob data that is rarely accessed. The Archive tier offers the lowest storage costs, but higher data retrieval costs and latency compared to the online tiers (Hot and Cool). Data must remain in the Archive tier for at least 180 days or be subject to an early deletion charge. For more information about the Archive tier, see [Archive access tier](#).

While a blob is in the Archive tier, it can't be read or modified. To read or download a blob in the Archive tier, you must first rehydrate it to an online tier, either Hot or Cool. Data in the Archive tier can take up to 15 hours to rehydrate, depending on the priority you specify for the rehydration operation. For more information about blob rehydration, see [Overview of blob rehydration from the Archive tier](#).

**Caution**

A blob in the Archive tier is offline. That is, it cannot be read or modified until it is rehydrated. The rehydration process can take several hours and has associated costs. Before you move data to the Archive tier, consider whether taking blob data offline may affect your workflows.

You can use the Azure portal, PowerShell, Azure CLI, or one of the Azure Storage client libraries to manage data archiving.

## Archive blobs on upload

To archive one ore more blob on upload, create the blob directly in the Archive tier.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [AzCopy](#)

To archive a blob or set of blobs on upload from the Azure portal, follow these steps:

1. Navigate to the target container.
2. Select the **Upload** button.
3. Select the file or files to upload.
4. Expand the **Advanced** section, and set the **Access tier** to *Archive*.
5. Select the **Upload** button.



## Advanced

### Authentication type

[Azure AD user account](#) [Account key](#)

### Blob type

[Block blob](#)

Upload .vhdx files as page blobs (recommended)

### Block size

[4 MB](#)

### Access tier

[Archive](#)

### Upload to folder

## Archive an existing blob

You can move an existing blob to the Archive tier in one of two ways:

- You can change a blob's tier with the [Set Blob Tier](#) operation. **Set Blob Tier** moves a single blob from one tier to another.

Keep in mind that when you move a blob to the Archive tier with **Set Blob Tier**, then you can't read or modify the blob's data until you rehydrate the blob. If you may need to read or modify the blob's data before the early deletion interval has elapsed, then consider using a [Copy Blob](#) operation to create a copy of the blob in the Archive tier.

- You can copy a blob in an online tier to the Archive tier with the [Copy Blob](#) operation. You can call the **Copy Blob** operation to copy a blob from an online tier (Hot or Cool) to the Archive tier. The source blob remains in the online tier, and you can continue to read or modify its data in the online tier.

### Archive an existing blob by changing its tier

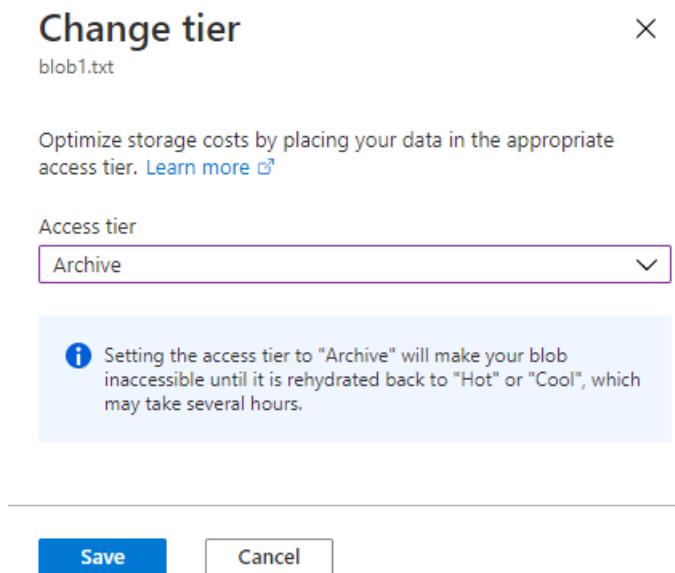
Use the **Set Blob Tier** operation to move a blob from the Hot or Cool tier to the Archive tier. The **Set Blob Tier** operation is best for scenarios where you won't need to access the archived data before the early deletion interval has elapsed.

The **Set Blob Tier** operation changes the tier of a single blob. To move a set of blobs to the Archive tier with optimum performance, Microsoft recommends performing a bulk archive operation. The bulk archive operation sends a batch of **Set Blob Tier** calls to the service in a single transaction. For more information, see [Bulk archive](#).

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [AzCopy](#)

To move an existing blob to the Archive tier in the Azure portal, follow these steps:

1. Navigate to the blob's container.
2. Select the blob to archive.
3. Select the **Change tier** button.
4. Select *Archive* from the **Access tier** dropdown.
5. Select **Save**.



## Archive an existing blob with a copy operation

Use the [Copy Blob](#) operation to copy a blob from the Hot or Cool tier to the Archive tier. The source blob remains in the Hot or Cool tier, while the destination blob is created in the Archive tier.

A [Copy Blob](#) operation is best for scenarios where you may need to read or modify the archived data before the early deletion interval has elapsed. You can access the source blob's data without needing to rehydrate the archived blob.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [AzCopy](#)

N/A

## Bulk archive

When moving a large number of blobs to the Archive tier, use a batch operation for optimal performance. A batch operation sends multiple API calls to the service with a single request. The suboperations supported by the [Blob Batch](#) operation include [Delete Blob](#) and [Set Blob Tier](#).

To archive blobs with a batch operation, use one of the Azure Storage client libraries. The following code example shows how to perform a basic batch operation with the .NET client library:

```

static async Task BulkArchiveContainerContents(string accountName, string containerName)
{
 string containerUri = string.Format("https://{}.{}/blob.core.windows.net/{}", accountName, containerName);

 // Get container client, using Azure AD credentials.
 BlobUriBuilder containerUriBuilder = new BlobUriBuilder(new Uri(containerUri));
 BlobContainerClient blobContainerClient = new BlobContainerClient(containerUriBuilder.ToUri(), new DefaultAzureCredential());

 // Get URIs for blobs in this container and add to stack.
 var uris = new Stack<Uri>();
 await foreach (var item in blobContainerClient.GetBlobsAsync())
 {
 uris.Push(blobContainerClient.GetBlobClient(item.Name).Uri);
 }

 // Get the blob batch client.
 BlobBatchClient blobBatchClient = blobContainerClient.GetBlobBatchClient();

 try
 {
 // Perform the bulk operation to archive blobs.
 await blobBatchClient.SetBlobsAccessTierAsync(blobUrises: uris, accessTier: AccessTier.Archive);
 }
 catch (RequestFailedException e)
 {
 Console.WriteLine(e.Message);
 }
}

```

For an in-depth sample application that shows how to change tiers with a batch operation, see [AzBulkSetBlobTier](#).

## Use lifecycle management policies to archive blobs

You can optimize costs for blob data that is rarely accessed by creating lifecycle management policies that automatically move blobs to the Archive tier when they haven't been accessed or modified for a specified period of time. After you configure a lifecycle management policy, Azure Storage runs it once per day. For more information about lifecycle management policies, see [Optimize costs by automatically managing the data lifecycle](#).

You can use the Azure portal, PowerShell, Azure CLI, or an Azure Resource Manager template to create a lifecycle management policy. For simplicity, this section shows how to create a lifecycle management policy in the Azure portal only. For more examples showing how to create lifecycle management policies, see [Configure a lifecycle management policy](#).

**Caution**

Before you use a lifecycle management policy to move data to the Archive tier, verify that that data does not need to be deleted or moved to another tier for at least 180 days. Data that is deleted or moved to a different tier before the 180 day period has elapsed is subject to an early deletion fee.

Also keep in mind that data in the Archive tier must be rehydrated before it can be read or modified. Rehydrating a blob from the Archive tier can take several hours and has associated costs.

To create a lifecycle management policy to archive blobs in the Azure portal, follow these steps:

1. Navigate to your storage account in the portal.
2. Under **Data management**, locate the **Lifecycle management** settings.

3. Select the **Add a rule** button.
4. On the **Details** tab, specify a name for your rule.
5. Specify the rule scope: either **Apply rule to all blobs in your storage account**, or **Limit blobs with filters**.
6. Select the types of blobs for which the rule is to be applied, and specify whether to include blob snapshots or versions.

## Add a rule ...

**1** Details

**2** Base blobs

A rule is made up of one or more conditions and actions that apply to the entire storage account. Optionally, specify that rules will apply to particular blobs by limiting with filters.

Rule name \*

sample-archive-rule

Rule scope \*

- Apply rule to all blobs in your storage account  
 Limit blobs with filters

Blob type \*

- Block blobs  
 Append blobs

Blob subtype \*

- Base blobs  
 Snapshots  
 Versions

Previous

Next

7. Depending on your selections, you can configure rules for base blobs (current versions), previous versions, or blob snapshots. Specify one of two conditions to check for:

- Objects were last modified some number of days ago.
- Objects were last accessed some number of days ago.

Only one of these conditions can be applied to move a particular type of object to the Archive tier per rule. For example, if you define an action that archives base blobs if they haven't been modified for 90 days, then you can't also define an action that archives base blobs if they haven't been accessed for 90 days. Similarly, you can define one action per rule with either of these conditions to archive previous versions, and one to archive snapshots.

8. Next, specify the number of days to elapse after the object is modified or accessed.

9. Specify that the object is to be moved to the Archive tier after the interval has elapsed.

## Add a rule

Details     Base blobs

Lifecycle management uses your rules to automatically move blobs to cooler tiers or to delete them. If you create multiple rules, the associated actions must be implemented in tier order (from hot to cool storage, then archive, then deletion).

The screenshot shows the 'Base blobs' configuration section of the Azure Storage Lifecycle Management rule editor. It is divided into two main sections: 'If' and 'Then'.

**If:** This section contains a condition and a value. The condition is "Base blobs were" followed by a dropdown menu with two options: "Last modified" (unchecked) and "Last accessed" (checked). Below this is a text input field labeled "More than (days ago)" with the value "90".

**Then:** This section contains a single action: "Move to archive storage". Below this action is a warning message: "⚠ If you have workloads that require real-time read-access to these blobs, moving them to archive is not recommended. Blobs in archive must first be rehydrated to hot or cool to read them. [Learn more](#)".

At the bottom of the configuration area are two buttons: "Previous" and "Add".

10. If you chose to limit the blobs affected by the rule with filters, you can specify a filter, either with a blob prefix or blob index match.

11. Select the **Add** button to add the rule to the policy.

After you create the lifecycle management policy, you can view the JSON for the policy on the **Lifecycle management** page by switching from **List view** to **Code view**.

Here's the JSON for the simple lifecycle management policy created in the images shown above:

```
{
 "rules": [
 {
 "enabled": true,
 "name": "sample-archive-rule",
 "type": "Lifecycle",
 "definition": {
 "actions": {
 "baseBlob": {
 "tierToArchive": {
 "daysAfterLastAccessTimeGreater Than": 90
 }
 }
 },
 "filters": {
 "blobTypes": [
 "blockBlob"
]
 }
 }
 }
]
}
```

## See also

- [Hot, Cool, and Archive access tiers for blob data](#)
- [Blob rehydration from the Archive tier](#)
- [Rehydrate an archived blob to an online tier](#)

# Rehydrate an archived blob to an online tier

8/22/2022 • 16 minutes to read • [Edit Online](#)

To read a blob that is in the Archive tier, you must first rehydrate the blob to an online tier (Hot or Cool) tier. You can rehydrate a blob in one of two ways:

- By copying it to a new blob in the Hot or Cool tier with the [Copy Blob](#) operation. Microsoft recommends this option for most scenarios.
- By changing its tier from Archive to Hot or Cool with the [Set Blob Tier](#) operation.

When you rehydrate a blob, you can specify the priority for the operation to either standard priority or high priority. A standard-priority rehydration operation may take up to 15 hours to complete. A high-priority operation is prioritized over standard-priority requests and may complete in less than one hour for objects under 10 GB in size. You can change the rehydration priority from *Standard* to *High* while the operation is pending.

You can configure Azure Event Grid to fire an event when rehydration is complete and run application code in response. To learn how to handle an event that runs an Azure Function when the blob rehydration operation is complete, see [Run an Azure Function in response to a blob rehydration event](#).

For more information about rehydrating a blob, see [Blob rehydration from the Archive tier](#).

## Rehydrate a blob with a copy operation

To rehydrate a blob from the Archive tier by copying it to an online tier, use PowerShell, Azure CLI, or one of the Azure Storage client libraries. Keep in mind that when you copy an archived blob to an online tier, the source and destination blobs must have different names.

Copying an archived blob to an online destination tier is supported within the same storage account. Beginning with service version 2021-02-12, you can copy an archived blob to a different storage account, as long as the destination account is in the same region as the source account.

After the copy operation is complete, the destination blob appears in the Archive tier. The destination blob is then rehydrated to the online tier that you specified in the copy operation. When the destination blob is fully rehydrated, it becomes available in the new online tier.

### Rehydrate a blob to the same storage account

The following examples show how to copy an archived blob to a blob in the Hot tier in the same storage account.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [AzCopy](#)

N/A

### Rehydrate a blob to a different storage account in the same region

The following examples show how to copy an archived blob to a blob in the Hot tier in a different storage account.

- [Portal](#)

- [PowerShell](#)
- [Azure CLI](#)
- [AzCopy](#)

N/A

#### **Rehydrate from a secondary region**

If you've configured your storage account to use read-access geo-redundant storage (RA-GRS), then you can copy an archived blob that is located in a secondary region to an online tier in a different storage account that is located in that same secondary region.

To rehydrate from a secondary region, use the same guidance that is presented in the previous section ([Rehydrate a blob to a different storage account in the same region](#)). Append the suffix `-secondary` to the account name of the source endpoint. For example, if your primary endpoint for Blob storage is `myaccount.blob.core.windows.net`, then the secondary endpoint is `myaccount-secondary.blob.core.windows.net`. The account access keys for your storage account are the same for both the primary and secondary endpoints.

To learn more about obtaining read access to secondary regions, see [Read access to data in the secondary region](#).

## Rehydrate a blob by changing its tier

To rehydrate a blob by changing its tier from Archive to Hot or Cool, use the Azure portal, PowerShell, or Azure CLI.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [AzCopy](#)

To change a blob's tier from Archive to Hot or Cool in the Azure portal, follow these steps:

1. Locate the blob to rehydrate in the Azure portal.
2. Select the **More** button on the right side of the page.
3. Select **Change tier**.
4. Select the target access tier from the **Access tier** dropdown.
5. From the **Rehydrate priority** dropdown, select the desired rehydration priority. Keep in mind that setting the rehydration priority to *High* typically results in a faster rehydration, but also incurs a greater cost.

The screenshot shows the 'Change tier' dialog for a blob named 'blob5.txt'. It includes fields for 'Access tier' (set to 'Cool') and 'Rehydrate priority' (set to 'Standard'). A note at the bottom states: 'Rehydrating a blob from Archive to Hot or Cool may take several hours to complete.' At the bottom are 'Save' and 'Cancel' buttons.

6. Select the **Save** button.

## Bulk rehydrate a set of blobs

To rehydrate a large number of blobs at one time, call the [Blob Batch](#) operation to call [Set Blob Tier](#) as a bulk operation. For a code example that shows how to perform the batch operation, see [AzBulkSetBlobTier](#).

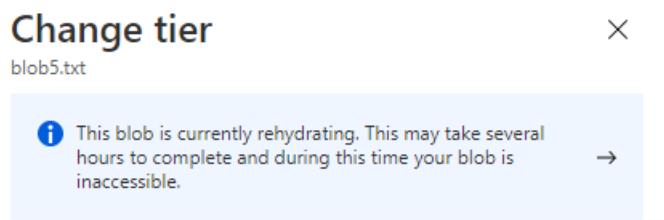
## Check the status of a rehydration operation

While the blob is rehydrating, you can check its status and rehydration priority using the Azure portal, PowerShell, or Azure CLI. The status property may return *rehydrate-pending-to-hot* or *rehydrate-pending-to-cool*, depending on the target tier for the rehydration operation. The rehydration priority property returns either *Standard* or *High*.

Keep in mind that rehydration of an archived blob may take up to 15 hours, and repeatedly polling the blob's status to determine whether rehydration is complete is inefficient. Using Azure Event Grid to capture the event that fires when rehydration is complete offers better performance and cost optimization. To learn how to run an Azure Function when an event fires on blob rehydration, see [Run an Azure Function in response to a blob rehydration event](#).

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [AzCopy](#)

To check the status and priority of a pending rehydration operation in the Azure portal, display the **Change tier** dialog for the blob:



When the rehydration is complete, you can see in the Azure portal that the fully rehydrated blob now appears in the targeted online tier.

Name	Modified	Access tier	Blob type	Size	Lease state	...
<input type="checkbox"/> blob1.txt	8/10/2021, 5:29:50 PM	Hot (Inferred)	Block blob	48 B	Available	...
<input type="checkbox"/> blob2.txt	8/2/2021, 4:18:02 PM	Cool	Block blob	20 B	Available	...
<input type="checkbox"/> blob3.txt	8/2/2021, 5:22:57 PM	Archive	Block blob	9 B	Available	...
<input type="checkbox"/> blob4.txt	8/2/2021, 8:51:51 PM	Hot	Block blob	9 B	Available	...
<input type="checkbox"/> blob5.txt	8/2/2021, 10:18:48 PM	Cool	Block blob	9 B	Available	...
<input type="checkbox"/> blob6.txt	8/3/2021, 8:35:43 AM	Hot	Block blob	9 B	Available	...
<input type="checkbox"/> blob7.txt	8/3/2021, 8:35:43 AM	Archive	Block blob	9 B	Available	...
<input type="checkbox"/> blob8.txt	8/3/2021, 8:44:59 AM	Archive	Block blob	9 B	Available	...

## Change the rehydration priority of a pending operation

While a standard-priority rehydration operation is pending, you can change the rehydration priority setting for a blob from *Standard* to *High* to rehydrate that blob more quickly.

The rehydration priority setting can't be lowered from *High* to *Standard* for a pending operation. Also keep in mind that changing the rehydration priority may have a billing impact. For more information, see [Blob rehydration from the Archive tier](#).

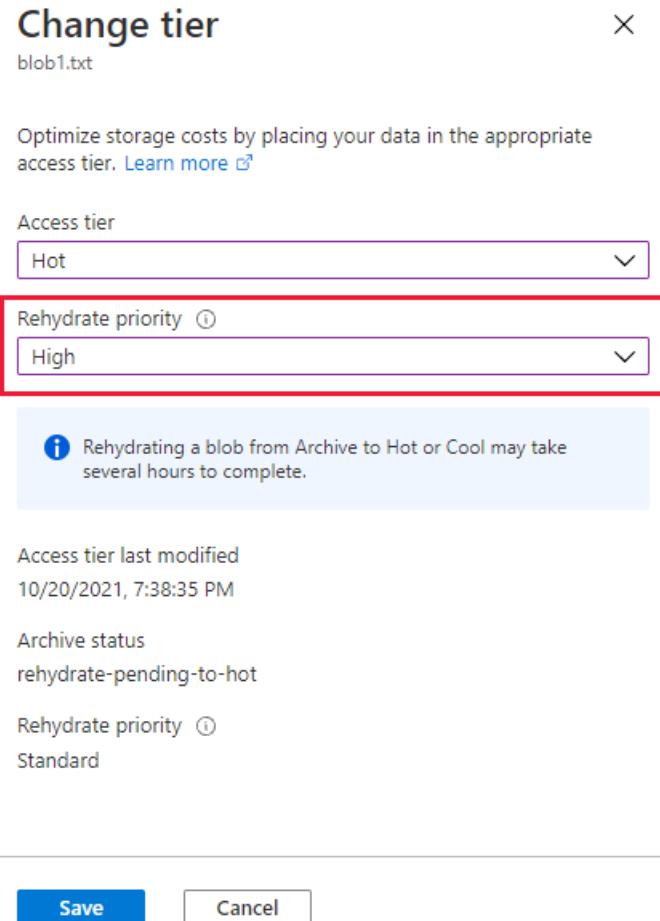
### Change the rehydration priority for a pending Set Blob Tier operation

To change the rehydration priority while a standard-priority [Set Blob Tier](#) operation is pending, use the Azure portal, PowerShell, Azure CLI, or one of the Azure Storage client libraries.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [AzCopy](#)

To change the rehydration priority for a pending operation with the Azure portal, follow these steps:

1. Navigate to the blob for which you want to change the rehydration priority, and select the blob.
2. Select the **Change tier** button.
3. In the **Change tier** dialog, set the access tier to the target online access tier for the rehydrating blob (Hot or Cool). The **Archive status** field shows the target online tier.
4. In the **Rehydrate priority** dropdown, set the priority to *High*.
5. Select **Save**.



### Change the rehydration priority for a pending Copy Blob operation

When you rehydrate a blob by copying the archived blob to an online tier, Azure Storage immediately creates the destination blob in the Archive tier. The destination blob is then rehydrated to the target tier with the priority specified on the copy operation. For more information on rehydrating an archived blob with a copy operation, see [Copy an archived blob to an online tier](#).

To perform the copy operation from the Archive tier to an online tier with Standard priority, use PowerShell, Azure CLI, or one of the Azure Storage client libraries. For more information, see [Rehydrate a blob with a copy operation](#). Next, to change the rehydration priority from *Standard* to *High* for the pending rehydration, call **Set Blob Tier** on the destination blob and specify the target tier.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [AzCopy](#)

After you've initiated the copy operation, you'll see in the Azure portal, that both the source and destination blob are in the Archive tier. The destination blob is rehydrating with Standard priority.

## blob1-rehydrated.txt ...

Blob

Save Discard Download Refresh | Delete | Change tier Acquire lease Break lease

This blob is currently rehydrating (Standard priority) and can't be downloaded. [Learn more](#)

[Overview](#) [Versions](#) [Snapshots](#) [Edit](#) [Generate SAS](#)

### Properties

URL	<a href="https://storagesamples...">https://storagesamples...</a>
LAST MODIFIED	10/26/2021, 12:21:01 PM
CREATION TIME	10/26/2021, 12:21:01 PM
VERSION ID	2021-10-26T19:21:01.9845338Z
TYPE	Block blob
SIZE	48 B
ACCESS TIER	Archive
ACCESS TIER LAST MODIFIED	10/26/2021, 12:21:01 PM
SERVER ENCRYPTED	true
ETAG	0x8D998B5BF8D1FCD
VERSION-LEVEL IMMUTABILITY POLICY	Disabled
CONTENT-TYPE	application/octet-stream
CONTENT-MD5	ncediF6hQtZnFTXes4X63g==
LEASE STATUS	Unlocked
LEASE STATE	Available
LEASE DURATION	-
COPY STATUS	Success
COPY COMPLETION TIME	10/26/2021, 12:21:01 PM
COPY SOURCE	<a href="https://storagesamples...">https://storagesamples...</a>

Undelete

To change the rehydration priority for the destination blob, follow these steps:

1. Select the destination blob.
2. Select the **Change tier** button.
3. In the **Change tier** dialog, set the access tier to the target online access tier for the rehydrating blob (Hot or Cool). The **Archive status** field shows the target online tier.
4. In the **Rehydrate priority** dropdown, set the priority to *High*.
5. Select **Save**.

The destination blob's properties page now shows that it's rehydrating with High priority.

blob1-rehydrated.txt ... X

Blob

Save Discard Download Refresh Delete Change tier Acquire lease Break lease

This blob is currently rehydrating (High priority) and can't be downloaded. [Learn more](#)

Overview Versions Snapshots Edit Generate SAS

Properties

URL	<a href="https://storagesamples...">https://storagesamples...</a> 
LAST MODIFIED	10/26/2021, 12:21:01 PM
CREATION TIME	10/26/2021, 12:21:01 PM
VERSION ID	2021-10-26T19:21:01.9845338Z
TYPE	Block blob
SIZE	48 B
ACCESS TIER	Archive
ACCESS TIER LAST MODIFIED	10/26/2021, 12:33:41 PM
SERVER ENCRYPTED	true
ETAG	0x8D998B5BF8D1FCD
VERSION-LEVEL IMMUTABILITY POLICY	Disabled
CONTENT-TYPE	application/octet-stream
CONTENT-MD5	ncediF6hQtZnFTXes4X63g==
LEASE STATUS	Unlocked
LEASE STATE	Available
LEASE DURATION	-
COPY STATUS	Success
COPY COMPLETION TIME	10/26/2021, 12:21:01 PM
COPY SOURCE	<a href="https://storagesamples...">https://storagesamples...</a> 

[Undelete](#)

## See also

- [Hot, Cool, and Archive access tiers for blob data.](#)
- [Overview of blob rehydration from the Archive tier](#)
- [Run an Azure Function in response to a blob rehydration event](#)
- [Reacting to Blob storage events](#)

# Run an Azure Function in response to a blob rehydration event

8/22/2022 • 12 minutes to read • [Edit Online](#)

To read a blob that is in the Archive tier, you must first rehydrate the blob to the Hot or Cool tier. The rehydration process can take several hours to complete. Instead of repeatedly polling the status of the rehydration operation, you can configure [Azure Event Grid](#) to fire an event when the blob rehydration operation is complete and handle this event in your application.

When an event occurs, Event Grid sends the event to an event handler via an endpoint. A number of Azure services can serve as event handlers, including [Azure Functions](#). An Azure Function is a block of code that can execute in response to an event. This how-to walks you through the process of developing an Azure Function and then configuring Event Grid to run the function in response to an event that occurs when a blob is rehydrated.

This article shows you how to create and test an Azure Function with .NET from Visual Studio. You can build Azure Functions from a variety of local development environments and using a variety of different programming languages. For more information about supported languages for Azure Functions, see [Supported languages in Azure Functions](#). For more information about development options for Azure Functions, see [Code and test Azure Functions locally](#).

For more information about rehydrating blobs from the Archive tier, see [Overview of blob rehydration from the Archive tier](#).

## Prerequisites

This article shows you how to use [Visual Studio 2019](#) or later to develop an Azure Function with .NET. You can install Visual Studio Community for free. Make sure that you [configure Visual Studio for Azure Development with .NET](#).

To debug the Azure Function locally, you will need to use a tool that can send an HTTP request, such as Postman.

An [Azure subscription](#) is required. If you don't already have an account, [create a free one](#) before you begin.

## Create an Azure Function app

A function app is an Azure resource that serves as a container for your Azure Functions. You can use a new or existing function app to complete the steps described in this article.

To create a new function app in the Azure portal, follow these steps:

1. In the Azure portal, search for *Function App*. Select the **Function App** icon to navigate to the list of function apps in your subscription.
2. Select the **Create** button to create a new function app.
3. On the **Basics** tab, specify a resource group, and provide a unique name for the new function app.
4. Make sure that the **Publish** option is set to *Code*.
5. From the **Runtime stack** dropdown, select *.NET*. The **Version** field is automatically populated to use the latest version of .NET core.

6. Select the region for the new function app.

Dashboard > Function App >

## Create Function App

**Basics**   **Hosting**   **Monitoring**   **Tags**   **Review + create**

Create a function app, which lets you group functions as a logical unit for easier management, deployment and sharing of resources. Functions lets you execute your code in a serverless environment without having to first create a VM or publish a web application.

**Project Details**

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

Subscription \*  ✓

Resource Group \*  ✓  
[Create new](#)

**Instance Details**

Function App name \*  ✓  
.azurewebsites.net

Publish \*  Code  Docker Container

Runtime stack \*  ✓

Version \*  ✓

Region \*  ✓

**Review + create**   [< Previous](#)   [Next : Hosting >](#)

7. After you have completed the **Basics** tab, navigate to the **Hosting** tab.
8. On the **Hosting** tab, select the storage account where your Azure Function will be stored. You can choose an existing storage account or create a new one.
9. Make sure that the **Operating system** field is set to *Windows*.
10. In the **Plan type** field, select *Consumption (Serverless)*. For more information about this plan, see [Azure Functions Consumption plan hosting](#).

## Create Function App



Basics    **Hosting**    Monitoring    Tags    Review + create

### Storage

When creating a function app, you must create or link to a general-purpose Azure Storage account that supports Blobs, Queue, and Table storage.

Storage account \*

storagesamplesfunctions (v2)

[Create new](#)



### Operating system

The Operating System has been recommended for you based on your selection of runtime stack.

Operating System \*

Linux  Windows

### Plan

The plan you choose dictates how your app scales, what features are enabled, and how it is priced. [Learn more](#)

Plan type \*

Consumption (Serverless)



[Review + create](#)

[< Previous](#)

[Next : Monitoring >](#)

11. Select **Review + Create** to create the new function app.

To learn more about configuring your function app, see [Manage your function app](#) in the Azure Functions documentation.

## Create an Azure Function as an Event Grid trigger

Next, create an Azure Function that will run when a blob is rehydrated in a particular storage account. Follow these steps to create an Azure Function in Visual Studio with C# and .NET Core:

1. Launch Visual Studio 2019, and create a new Azure Functions project. For details, follow the instructions described in [Create a function app project](#).
2. On the **Create a new Azure Functions application** step, select the following values:
  - By default, the Azure Functions runtime is set to **Azure Functions v3 (.NET Core)**. Microsoft recommends using this version of the Azure Functions runtime.
  - From the list of possible triggers, select **Event Grid Trigger**. For more information on why an Event Grid trigger is the recommended type of trigger for handling a Blob Storage event with an Azure Function, see [Use a function as an event handler for Event Grid events](#).
  - The **Storage Account** setting indicates where your Azure Function will be stored. You can select an existing storage account or create a new one.
3. Select **Create** to create the new project in Visual Studio.
4. Next, rename the class and Azure Function, as described in [Rename the function](#). Choose a name that's appropriate for your scenario.
5. In Visual Studio, select **Tools | NuGet Package Manager | Package Manager Console**, and then install the following packages from the console:

```
Install-Package Azure.Storage.Blobs
Install-Package Microsoft.ApplicationInsights.WorkerService
Install-Package Microsoft.Azure.WebJobs.Logging.ApplicationInsights
```

6. In the class file for your Azure Function, paste in the following using statements:

```
using System;
using System.IO;
using System.Text;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.EventGrid.Models;
using Microsoft.Azure.WebJobs.Extensions.EventGrid;
using Microsoft.Extensions.Logging;
using Azure;
using Azure.Storage.Blobs;
using Azure.Storage.Blobs.Models;
```

7. Locate the **Run** method in the class file. This is the method that runs when an event occurs. Paste the following code into the body of the **Run** method. Remember to replace placeholder values in angle brackets with your own values:

```

// When either Microsoft.Storage.BlobCreated or Microsoft.Storage.BlobTierChanged
// event occurs, write the event details to a log blob in the same container
// as the event subject (the blob for which the event occurred).

// Create a unique name for the log blob.
string logBlobName = string.Format("function-log-{0}.txt", DateTime.UtcNow.Ticks);

// Populate connection string with your Shared Key credentials.
const string ConnectionString = "DefaultEndpointsProtocol=https;AccountName=<account-name>;AccountKey=<account-key>;EndpointSuffix=core.windows.net";

// Get data from the event.
dynamic data = eventGridEvent.Data;
string eventBlobUrl = Convert.ToString(data.url);
string eventApi = Convert.ToString(data.api);

// Build string containing log information.
StringBuilder eventInfo = new StringBuilder();
eventInfo.AppendLine(string.Format("{0} operation occurred.", eventApi));
eventInfo.AppendLine(string.Format("Blob URL: {0}", eventBlobUrl));
eventInfo.AppendLine($"Additional event details:
 Id=[{eventGridEvent.Id}]
 EventType=[{eventGridEvent.EventType}]
 EventTime=[{eventGridEvent.EventTime}]
 Subject=[{eventGridEvent.Subject}]
 Topic=[{eventGridEvent.Topic}]");

// If event was BlobCreated and API call was CopyBlob, respond to the event.
bool copyBlobEventOccurred = (eventGridEvent.EventType == "Microsoft.Storage.BlobCreated") &&
 (eventApi == "CopyBlob");

// If event was BlobTierChanged and API call was SetBlobTier, respond to the event.
bool setTierEventOccurred = (eventGridEvent.EventType == "Microsoft.Storage.BlobTierChanged") &&
 (eventApi == "SetBlobTier");

// If one of these two events occurred, write event info to a log blob.
if (copyBlobEventOccurred | setTierEventOccurred)
{
 // Create log blob in same account and container.
 BlobUriBuilder logBlobUriBuilder = new BlobUriBuilder(new Uri(eventBlobUrl))
 {
 BlobName = logBlobName
 };

 BlobClient logBlobClient = new BlobClient(ConnectionString,
 logBlobUriBuilder.BlobContainerName,
 logBlobName);

 byte[] byteArray = Encoding.ASCII.GetBytes(eventInfo.ToString());

 try
 {
 // Write the log info to the blob.
 // Overwrite if the blob already exists.
 using (MemoryStream memoryStream = new MemoryStream(byteArray))
 {
 BlobContentInfo blobContentInfo =
 logBlobClient.Upload(memoryStream, overwrite: true);
 }
 }
 catch (RequestFailedException e)
 {
 Console.WriteLine(e.Message);
 throw;
 }
}
}

```

For more information on developing Azure Functions, see [Guidance for developing Azure Functions](#).

To learn more about the information that is included when a Blob Storage event is published to an event handler, see [Azure Blob Storage as Event Grid source](#).

## Run the Azure Function locally in the debugger

To test your Azure Function code locally, you need to manually send an HTTP request that triggers the event. You can post the request using a tool such as Postman.

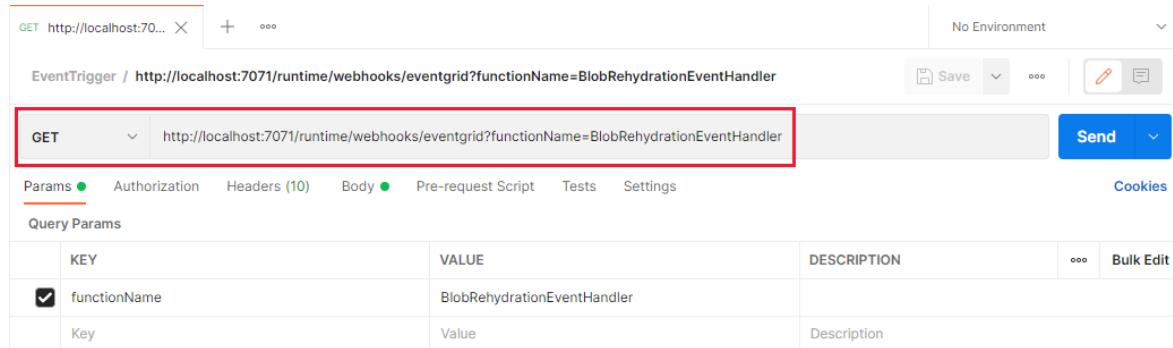
At the top of the class file for your Azure Function is a URL endpoint that you can use for testing in the local environment. Posting the request with this URL triggers the event in the local environment so that you can debug your code. The URL is in the following format:

```
http://localhost:7071/runtime/webhooks/EventGrid?functionName={functionname}
```

The request that you send to this endpoint is a simulated request. It does not send or receive data from your Azure Storage account.

Follow these steps to construct and send a request to this endpoint. This example shows how to send the request with Postman.

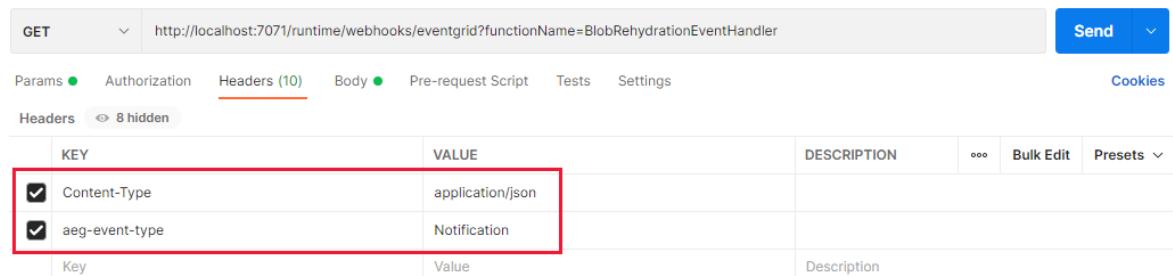
1. In Postman, create a new request.
2. Paste the URL shown above into the field for the request URL, substituting the name of your function for `{functionname}` and removing the curly braces. Make sure that the request verb is set to GET.



The screenshot shows the Postman interface with a GET request. The URL field contains `http://localhost:7071/runtime/webhooks/eventgrid?functionName=BlobRehydrationEventHandler`. The 'Params' tab is selected, showing a table with two rows: `functionName` with value `BlobRehydrationEventHandler` and `Key` with value `Description`. Other tabs include Authorization, Headers (10), Body, Pre-request Script, Tests, and Settings. A 'Send' button is visible on the right.

3. Add the `Content-Type` header and set it to `application/json`.

4. Add the `aeg-event-type` header and set it to `Notification`.



The screenshot shows the Postman interface with the same GET request. The 'Headers' tab is selected, showing a table with two rows: `Content-Type` with value `application/json` and `aeg-event-type` with value `Notification`. Other tabs include Params, Authorization, Body, Pre-request Script, Tests, and Settings. A 'Send' button is visible on the right.

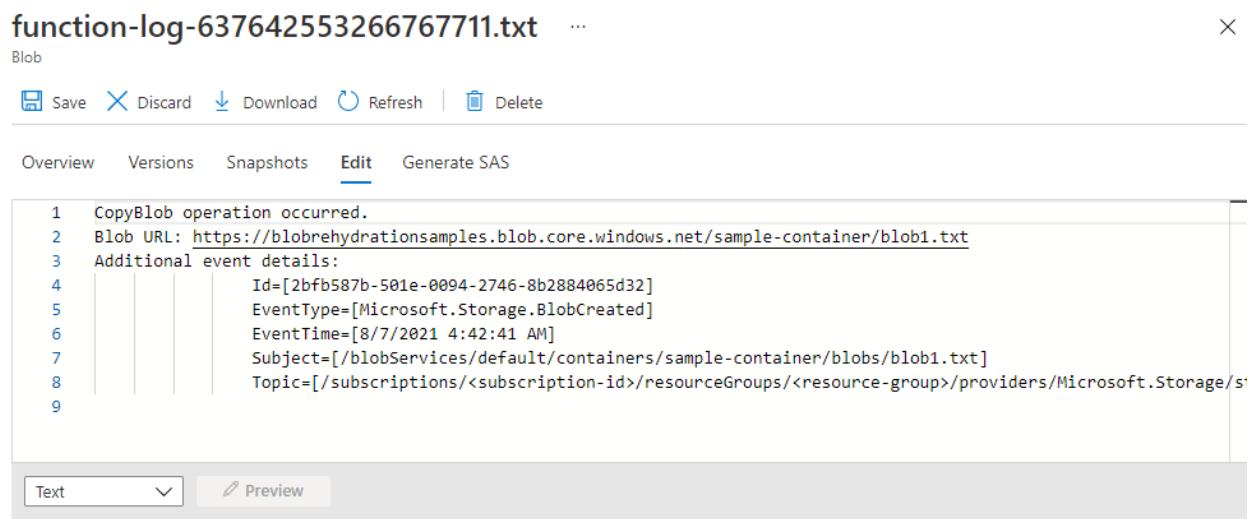
5. In Postman, specify the request body, with the body type set to `JSON` and the format to `raw`. The following example simulates a **Copy Blob** request. Replace placeholder values in angle brackets with your own values. Note that it is not necessary to change date/time or identifier values, because this is a simulated request:

```
[{"topic": "/subscriptions/<subscription-id>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>", "subject": "/blobServices/default/containers/<container-name>/blobs/<blob-name>", "eventType": "Microsoft.Storage.BlobCreated", "id": "2bfb587b-501e-0094-2746-8b2884065d32", "data": { "api": "CopyBlob", "clientRequestId": "3d4dedc7-6c27-4816-9405-fdbfa806b00c", "requestId": "2bfb587b-501e-0094-2746-8b2884000000", "eTag": "0x8D9595DCA505BDF", "contentType": "text/plain", "contentLength": 48, "blobType": "BlockBlob", "url": "https://<storage-account>.blob.core.windows.net/<container-name>/<blob-name>", "sequencer": "00000000000000000000000000000000201B00000000004092a5", "storageDiagnostics": { "batchId": "8a92736a-6006-0026-0046-8bd7f5000000" } }, "dataVersion": "", "metadataVersion": "1", "eventTime": "2021-08-07T04:42:41.0730463Z"}]
```

6. In Visual Studio, place any desired breakpoints in your code, and press F5 to run the debugger.
  7. In Postman, select the **Send** button to send the request to the endpoint.

When you send the request, Event Grid calls your Azure Function, and you can debug it normally. For additional information and examples, see [Manually post the request](#) in the Azure Functions documentation.

The request that triggers the event is simulated, but the Azure Function that runs when the event fires writes log information to a new blob in your storage account. You can verify the contents of the blob and view its last modified time in the Azure portal, as shown in the following image:



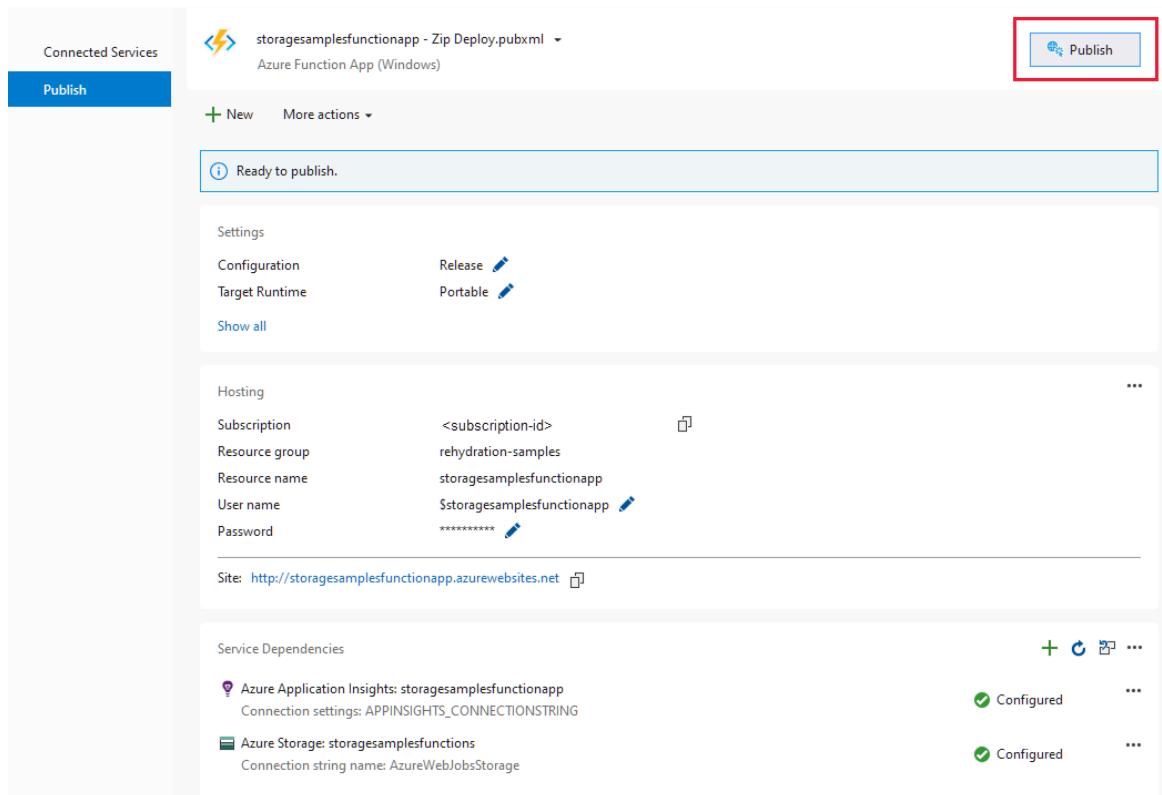
## Publish the Azure Function

After you have tested your Azure Function locally, the next step is to publish the Azure Function to the Azure Function App that you created previously. The function must be published so that you can configure Event Grid to send events that happen on the storage account to the function endpoint.

Follow these steps to publish the function:

1. In Solution Explorer, select and hold (or right-click) your Azure Functions project and choose **Publish**.

2. In the **Publish** window, select **Azure** as the target, then choose **Next**.
3. Select **Azure Function App (Windows)** as the specific target, then choose **Next**.
4. On the **Functions instance** tab, select your subscription from the dropdown menu, then locate your Azure Function App in the list of available function apps.
5. Make sure that the **Run from package** file checkbox is selected.
6. Select **Finish** to prepare to publish the function.
7. On the **Publish** page, verify that the configuration is correct. If you see a warning that the service dependency to Application Insights is not configured, you can configure it from this page.
8. Select the **Publish** button to begin publishing the Azure Function to the Azure Function App that you created previously.



Whenever you make changes to the code in your Azure Function, you must publish the updated function to Azure.

## Subscribe to blob rehydration events from a storage account

You now have a function app that contains an Azure Function that can run in response to an event. The next step is to create an event subscription from your storage account. The event subscription configures the storage account to publish an event through Event Grid in response to an operation on a blob in your storage account. Event Grid then sends the event to the event handler endpoint that you've specified. In this case, the event handler is the Azure Function that you created in the previous section.

When you create the event subscription, you can filter which events are sent to the event handler. The events to capture when rehydrating a blob from the Archive tier are **Microsoft.Storage.BlobTierChanged**, corresponding to a [Set Blob Tier](#) operation, and **Microsoft.Storage.BlobCreated** events, corresponding to a [Copy Blob](#) operation. Depending on your scenario, you may want to handle only one of these events.

To create the event subscription, follow these steps:

1. In the Azure portal, navigate to the storage account that contains blobs to rehydrate from the Archive tier.

2. Select the **Events** setting in the left navigation pane.
3. On the **Events** page, select **More options**.
4. Select **Create Event Subscription**.
5. On the **Create Event Subscription** page, in the **Event subscription details** section, provide a name for the event subscription.
6. In the **Topic details** section, provide a name for the system topic. The system topic represents one or more events that are published by Azure Storage. For more information about system topics, see [System topics in Azure Event Grid](#).
7. In the **Event Types** section, select the **Blob Created** and **Blob Tier Changed** events. Depending on how you choose to rehydrate a blob from the Archive tier, one of these two events will fire.

Dashboard > blobrehydrationsamples >

## Create Event Subscription

Event Grid

**Basic**   [Filters](#)   [Additional Features](#)   [Delivery Properties](#)   [Advanced Editor](#)

Event Subscriptions listen for events emitted by the topic resource and send them to the endpoint resource. [Learn more](#)

**EVENT SUBSCRIPTION DETAILS**

Name \*  ✓

Event Schema  Blob Created    Blob Deleted    Directory Created    Directory Deleted    Blob Renamed    Directory Renamed    Blob Tier Changed    Blob Inventory Completed  

**TOPIC DETAILS**

Pick a topic resource for which events should be sent to this subscription.

Topic Type  

Source Resource  

System Topic Name \*  

**EVENT TYPES**

Pick which event types get pushed to your destination.

Filter to Event Types 2 selected

**ENDPOINT DETAILS**

Pick an event handler to receive your events. [Learn more](#)

Endpoint Type \*  

**Create**

8. In the **Endpoint details** section, select *Azure Function* from the dropdown menu.
9. Choose **Select an endpoint** to specify the function that you created in the previous section. In the **Select Azure Function** dialog, choose the subscription, resource group, and function app for your Azure Function. Finally, select the function name from the dropdown and choose **Confirm selection**.

## Select Azure Function

Event Grid

X

Subscription

Resource group

Function app \*

Slot \* ⓘ

Function \* ⓘ

**Confirm Selection**

10. Select the **Create** button to create the event subscription and begin sending events to the Azure Function event handler.

To learn more about event subscriptions, see [Azure Event Grid concepts](#).

## Test the Azure Function event handler

To test the Azure Function, you can trigger an event in the storage account that contains the event subscription. The event subscription that you created previously is filtering on two events, **Microsoft.Storage.BlobCreated** and **Microsoft.Storage.BlobTierChanged**. When either of these events fires, it will trigger your Azure Function.

The Azure Function shown in this article writes to a log blob in two scenarios:

- When the event is **Microsoft.Storage.BlobCreated** and the API operation is **Copy Blob**.
- When the event is **Microsoft.Storage.BlobTierChanged** and the API operation is **Set Blob Tier**.

To learn how to test the function by rehydrating a blob, see one of these two procedures:

- [Rehydrate a blob with a copy operation](#)
- [Rehydrate a blob by changing its tier](#)

After the rehydration is complete, the log blob is written to the same container as the blob that you rehydrated. For example, after you rehydrate a blob with a copy operation, you can see in the Azure portal that the original source blob remains in the Archive tier, the fully rehydrated destination blob appears in the targeted online tier, and the log blob that was created by the Azure Function also appears in the list.

Name	Modified	Access tier	Blob type	Size	Lease state	
blob1.txt	8/10/2021, 5:29:50 PM	Hot (Inferred)	Block blob	48 B	Available	...
blob2.txt	8/2/2021, 4:18:02 PM	Cool	Block blob	20 B	Available	...
blob3-copy.txt	8/10/2021, 6:02:28 PM	Hot	Block blob	9 B	Available	...
blob3.txt	8/2/2021, 5:22:57 PM	Archive	Block blob	9 B	Available	...
blob4.txt	8/2/2021, 8:51:51 PM	Hot	Block blob	9 B	Available	...
blob5.txt	8/2/2021, 10:18:48 PM	Archive	Block blob	9 B	Available	...
blob6.txt	8/3/2021, 8:35:43 AM	Hot	Block blob	9 B	Available	...
blob7.txt	8/3/2021, 8:35:43 AM	Archive	Block blob	9 B	Available	...
blob8.txt	8/3/2021, 8:44:59 AM	Archive	Block blob	9 B	Available	...
function-log-637642421576133435.txt	8/10/2021, 6:29:17 PM	Hot (Inferred)	Block blob	603 B	Available	...

Keep in mind that rehydrating a blob can take up to 15 hours, depending on the rehydration priority setting. If you set the rehydration priority to **High**, rehydration may complete in under one hour for blobs that are less than 10 GB in size. However, a high-priority rehydration incurs a greater cost. For more information, see [Overview of blob rehydration from the Archive tier](#).

#### TIP

Although the goal of this how-to is to handle these events in the context of blob rehydration, for testing purposes it may also be helpful to observe these events in response to uploading a blob or changing an online blob's tier (*i.e.*, from Hot to Cool), because the event fires immediately.

For more information on how to filter events in Event Grid, see [How to filter events for Azure Event Grid](#).

## See also

- [Hot, Cool, and Archive access tiers for blob data](#)
- [Overview of blob rehydration from the Archive tier](#)
- [Rehydrate an archived blob to an online tier](#)
- [Reacting to Blob storage events](#)

# Configure a lifecycle management policy

8/22/2022 • 6 minutes to read • [Edit Online](#)

Azure Storage lifecycle management offers a rule-based policy that you can use to transition blob data to the appropriate access tiers or to expire data at the end of the data lifecycle. A lifecycle policy acts on a base blob, and optionally on the blob's versions or snapshots. For more information about lifecycle management policies, see [Optimize costs by automatically managing the data lifecycle](#).

A lifecycle management policy is comprised of one or more rules that define a set of actions to take based on a condition being met. For a base blob, you can choose to check one of two conditions:

- The number of days since the blob was last modified.
- The number of days since the blob was last accessed. To use this condition in an action, you must first [optionally enable access time tracking](#).

When the selected condition is true, then the management policy performs the specified action. For example, if you have defined an action to move a blob from the hot tier to the cool tier if it has not been modified for 30 days, then the lifecycle management policy will move the blob 30 days after the last write operation to that blob.

For a blob snapshot or version, the condition that is checked is the number of days since the snapshot or version was created.

## Optionally enable access time tracking

Before you configure a lifecycle management policy, you can choose to enable blob access time tracking. When access time tracking is enabled, a lifecycle management policy can include an action based on the time that the blob was last accessed with a read or write operation.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Template](#)

To enable last access time tracking with the Azure portal, follow these steps:

1. Navigate to your storage account in the Azure portal.
2. In the **Data management** section, select **Lifecycle management**.

The screenshot shows the 'Lifecycle management' blade in the Azure portal. The left sidebar lists 'Object replication', 'Blob inventory', 'Static website', 'Lifecycle management' (which is selected), and 'Azure search'. The main area displays a policy named 'move-log-to-cool-after-30-days' with the 'Enable access tracking' checkbox checked. The policy table shows one rule: 'move-log-to-cool-after-30-days' is Enabled and of type Block.

Name	Status	Blob type
move-log-to-cool-after-30-days	Enabled	Block

Use the `daysAfterLastAccessTimeGreater Than` property to specify the number of days from last access after which an action should be taken on a blob.

## Create or manage a policy

You can add, edit, or remove a lifecycle management policy with the Azure portal, PowerShell, Azure CLI, or an Azure Resource Manager template.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Template](#)

There are two ways to add a policy through the Azure portal.

- [List view](#)
- [Code view](#)

### List view

1. In the Azure portal, navigate to your storage account.
2. Under **Data management**, select **Lifecycle Management** to view or change lifecycle management policies.
3. Select the **List View** tab.
4. Select **Add a rule** and name your rule on the **Details** form. You can also set the **Rule scope**, **Blob type**, and **Blob subtype** values. The following example sets the scope to filter blobs. This causes the **Filter set** tab to be added.

[Dashboard](#) > [lifecyclesamples](#) >

**Add a rule** ... X

1 Details   2 Base blobs   3 Filter set

A rule is made up of one or more conditions and actions that apply to the entire storage account. Optionally, specify that rules will apply to particular blobs by limiting with filters.

Rule name \*

Rule scope \*  
 Apply rule to all blobs in your storage account  
 Limit blobs with filters

Blob type \*  
 Block blobs  
 Append blobs

Blob subtype \*  
 Base blobs  
 Snapshots  
 Versions

---

[Previous](#) [Next](#)

5. Select **Base blobs** to set the conditions for your rule. In the following example, blobs are moved to cool storage if they haven't been modified for 30 days.

Dashboard > lifecyclesamples >

## Add a rule

**If**

Base blobs were \*

Last modified  
 Last accessed

More than (days ago) \*

30

**Then**

Move to cool storage

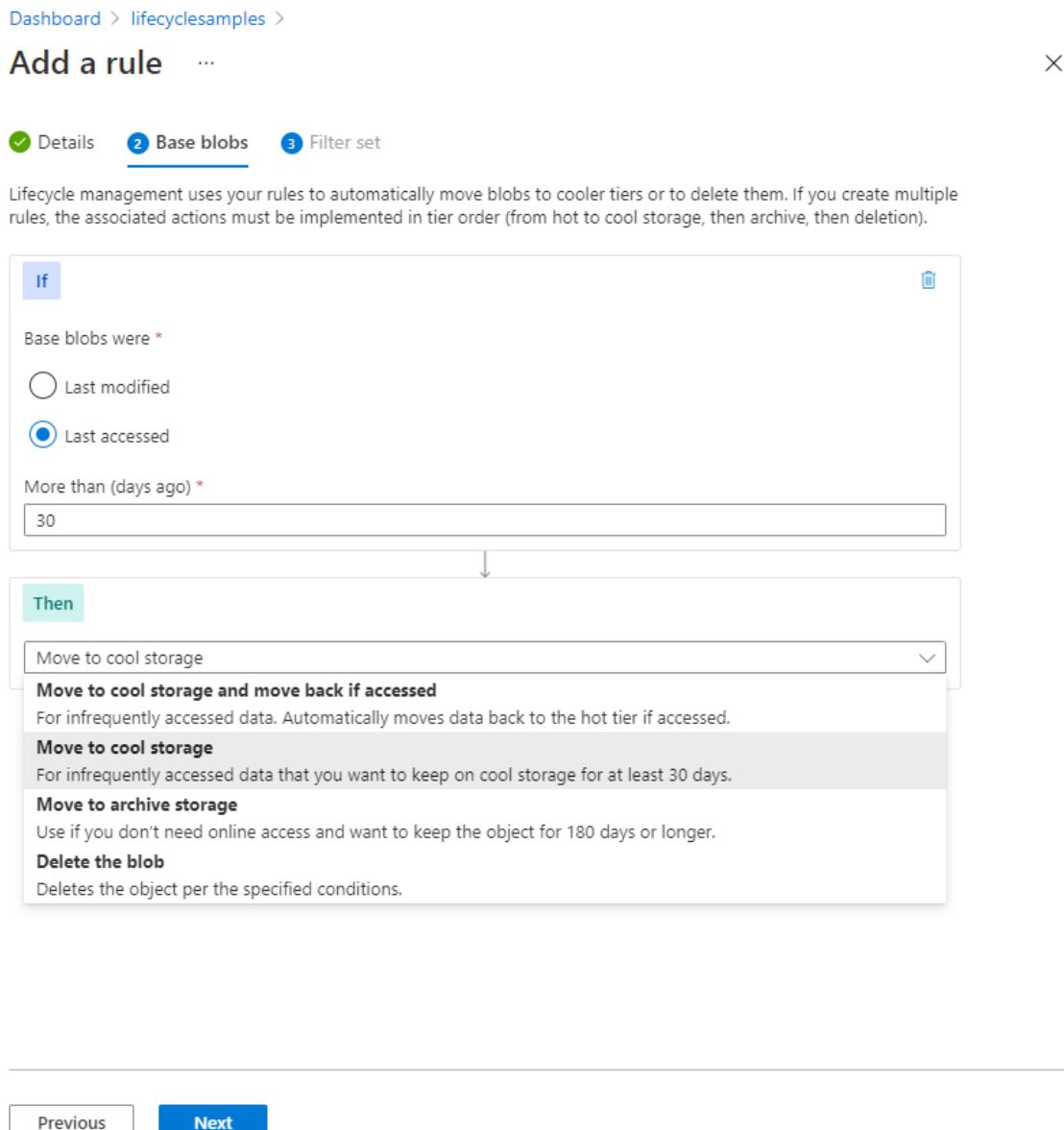
**Move to cool storage and move back if accessed**  
For infrequently accessed data. Automatically moves data back to the hot tier if accessed.

**Move to cool storage**  
For infrequently accessed data that you want to keep on cool storage for at least 30 days.

**Move to archive storage**  
Use if you don't need online access and want to keep the object for 180 days or longer.

**Delete the blob**  
Deletes the object per the specified conditions.

Previous    **Next**



The **Last accessed** option is available only if you have enabled access time tracking. To learn how to enable access tracking, see [Optionally enable access time tracking](#).

6. If you selected **Limit blobs with filters** on the **Details** page, select **Filter set** to add an optional filter. The following example filters on blobs whose name begins with */log* in a container called *sample-container*.

## Add a rule

X

Details     Base blobs     Filter set

### Blob prefix

Filter blobs by name or first letters. To find items in a specific container, enter the name of the container followed by a forward slash, then the blob name or first letters. For example, to show all blobs starting with "a", type: "mycontainer/a".

### Blob prefix

\*

### Blob index match

If you have indexed items in containers with keys and values, you can filter for them.

Key	Value
-----	-------

<input type="text" value="Enter an index key"/>	<input type="button" value="=="/>	<input type="text" value="Enter a value"/>
-------------------------------------------------	-----------------------------------	--------------------------------------------

[Previous](#)

**Add**

7. Select **Add** to add the new policy.

### Code view

1. In the Azure portal, navigate to your storage account.
2. Under **Data management**, select **Lifecycle Management** to view or change lifecycle management policies.
3. Select the **Code View** tab. On this tab, you can define a lifecycle management policy in JSON.

The following sample JSON defines a lifecycle 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.

```
{
 "rules": [
 {
 "enabled": true,
 "name": "move-to-cool",
 "type": "Lifecycle",
 "definition": {
 "actions": {
 "baseBlob": {
 "tierToCool": {
 "daysAfterModificationGreaterThan": 30
 }
 }
 },
 "filters": {
 "blobTypes": [
 "blockBlob"
],
 "prefixMatch": [
 "sample-container/log"
]
 }
 }
 }
]
}
```

A lifecycle management policy must be read or written in full. Partial updates are not supported.

**NOTE**

Each rule can have up to 10 case-sensitive prefixes and up to 10 blob index tag conditions.

**NOTE**

If you enable firewall rules for your storage account, lifecycle management requests may be blocked. You can unblock these requests by providing exceptions for trusted Microsoft services. For more information, see the **Exceptions** section in [Configure firewalls and virtual networks](#).

## See also

- [Optimize costs by automatically managing the data lifecycle](#)
- [Hot, Cool, and Archive access tiers for blob data](#)

# Configure object replication for block blobs

8/22/2022 • 12 minutes to read • [Edit Online](#)

Object replication asynchronously copies block blobs between a source storage account and a destination account. When you configure object replication, you create a replication policy that specifies the source storage account and the destination account. A replication policy includes one or more rules that specify a source container and a destination container and indicate which block blobs in the source container will be replicated. For more information about object replication, see [Object replication for block blobs](#).

This article describes how to configure an object replication policy by using the Azure portal, PowerShell, or Azure CLI. You can also use one of the Azure Storage resource provider client libraries to configure object replication.

## Prerequisites

Before you configure object replication, create the source and destination storage accounts if they don't already exist. The source and destination accounts can be either general-purpose v2 storage accounts or premium block blob accounts. For more information, see [Create an Azure Storage account](#).

Object replication requires that blob versioning is enabled for both the source and destination account, and that blob change feed is enabled for the source account. To learn more about blob versioning, see [Blob versioning](#). To learn more about change feed, see [Change feed support in Azure Blob Storage](#). Keep in mind that enabling these features can result in additional costs.

To configure an object replication policy for a storage account, you must be assigned the Azure Resource Manager **Contributor** role, scoped to the level of the storage account or higher. For more information, see [Azure built-in roles](#) in the Azure role-based access control (Azure RBAC) documentation.

## Configure object replication with access to both storage accounts

If you have access to both the source and destination storage accounts, then you can configure the object replication policy on both accounts. The following examples show how to configure object replication with the Azure portal, PowerShell, or Azure CLI.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

When you configure object replication in the Azure portal, you only need to configure the policy on the source account. The Azure portal automatically creates the policy on the destination account after you configure it for the source account.

To create a replication policy in the Azure portal, follow these steps:

1. Navigate to the source storage account in the Azure portal.
2. Under **Data management**, select **Object replication**.
3. Select **Set up replication rules**.
4. Select the destination subscription and storage account.
5. In the **Container pairs** section, select a source container from the source account, and a destination

container from the destination account. You can create up to 10 container pairs per replication policy from the Azure portal. To configure more than 10 container pairs (up to 1000), see [Configure object replication using a JSON file](#).

The following image shows a set of replication rules.

The screenshot shows the 'Create replication rules for your organization' page. At the top, there's a note: 'When you create object replication rules, blob change feed and blob versioning are automatically enabled for the source and destination storage accounts. Enabling these features may increase costs.' Below this are sections for 'Destination details' and 'Container pair details'. In 'Destination details', 'Destination subscription' is set to '<destination-subscription>' and 'Destination storage account' is set to '<destination-account>'. In 'Container pair details', there's a table showing three source containers (src-container-1, src-container-2, src-container-3) mapped to destination containers (dest-container-1, dest-container-2, dest-container-3). Each row has a 'Copy over' column with options: 'Only new objects (change)', 'Everything (change)', or 'Custom (change)'. Below the table are dropdown menus for 'Select a source container' and 'Select a destination container'. At the bottom are 'Save and apply' and 'Cancel' buttons.

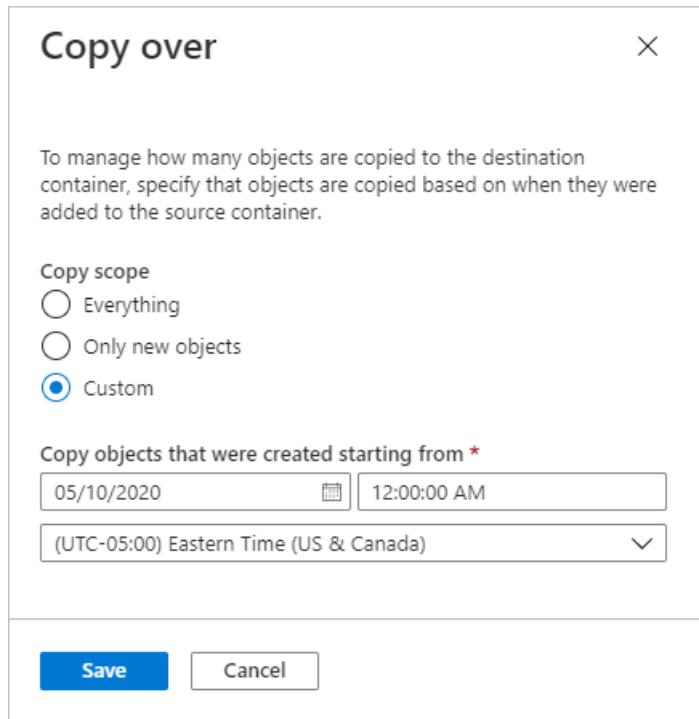
6. If desired, specify one or more filters to copy only blobs that match a prefix pattern. For example, if you specify a prefix **b**, only blobs whose name begin with that letter are replicated. You can specify a virtual directory as part of the prefix. You can add a maximum of up to five prefix matches. The prefix string doesn't support wildcard characters.

The following image shows filters that restrict which blobs are copied as part of a replication rule.

The screenshot shows the 'Add prefix matches' dialog box. It contains a descriptive text: 'A prefix match will find items like folders and blobs under the specified container that start with the specified input. For example, inputting "a" would filter any folders or blobs that start with "a". If multiple prefixes are specified, items that have any of these prefixes will be included.' Below this is a 'Prefix match' input field containing 'b', followed by two entries: 'sample-directory/b'. At the bottom are 'Save' and 'Cancel' buttons.

7. By default, the copy scope is set to copy only new objects. To copy all objects in the container or to copy objects starting from a custom date and time, select the **change** link and configure the copy scope for the container pair.

The following image shows a custom copy scope that copies objects from a specified date and time onward.



8. Select **Save and apply** to create the replication policy and start replicating data.

After you have configured object replication, the Azure portal displays the replication policy and rules, as shown in the following image.

The screenshot shows the 'objectreplicationsource | Object replication' page in the Azure portal. The left sidebar lists 'Data management' (Geo-replication, Data protection, Object replication, Blob inventory, Static website, Lifecycle management, Azure search), 'Settings' (Configuration, Resource sharing (CORS), Advisor recommendations, Endpoints, Locks), and 'Storage account'. The main area shows a summary of replication rules: 'When object replication is enabled, blobs are copied asynchronously from a source storage account to a destination account. Cross-tenant policies will appear under "Other accounts", along with policies on accounts not in an active subscription or on deleted accounts. The storage accounts may be in different Azure regions.' Below this are sections for 'Your accounts' and 'Other accounts'. Under 'Your accounts', there's a table for 'Objects copied from this account' with three rows:

Destination account	Source container	Destination container	Filters
objectreplicationsource	source-container1	dest-container1	1
objectreplicationsource	source-container2	dest-container2	0
objectreplicationsource	source-container3	dest-container3	0

Under 'Other accounts', there's a table for 'Objects copied into this account' with one row:

Source account	Source container	Destination container	Filters
No replication policies found			

## Configure object replication using a JSON file

If you don't have permissions to the source storage account or if you want to use more than 10 container pairs, then you can configure object replication on the destination account and provide a JSON file that contains the policy definition to another user to create the same policy on the source account. For example, if the source account is in a different Azure AD tenant from the destination account, then you can use this approach to configure object replication.

#### NOTE

Cross-tenant object replication is permitted by default for a storage account. To prevent replication across tenants, you can set the **AllowCrossTenantReplication** property (preview) to disallow cross-tenant object replication for your storage accounts. For more information, see [Prevent object replication across Azure Active Directory tenants](#).

The examples in this section show how to configure the object replication policy on the destination account, and then get the JSON file for that policy that another user can use to configure the policy on the source account.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To configure object replication on the destination account with a JSON file in the Azure portal, follow these steps:

1. Create a local JSON file that defines the replication policy on the destination account. Set the **policyId** field to *default* so that Azure Storage will define the policy ID.

An easy way to create a JSON file that defines a replication policy is to first create a test replication policy between two storage accounts in the Azure portal. You can then download the replication rules and modify the JSON file as needed.

2. Navigate to the **Object replication** settings for the destination account in the Azure portal.
3. Select **Upload replication rules**.
4. Upload the JSON file. The Azure portal displays the policy and rules that will be created, as shown in the following image.

The screenshot shows the 'Upload replication rules' page in the Azure portal. At the top, there's a breadcrumb navigation: Dashboard > centrallobjrepdest > Upload replication rules. On the right, there are 'Save' and 'Cancel' buttons. A note at the top says: 'When you create object replication rules, blob change feed and blob versioning are automatically enabled for the source storage account. Enabling these features may increase costs.' Below this is a note: 'Upload the file for replication rules below. Learn more about copying objects across tenants'. A warning message states: 'If you own the source account and you do not have a JSON file for replication rules, the owner of the destination may have it. If you own the destination account, replication will not start until the replication rules are downloaded and uploaded on the source account.' A 'File' input field contains the path: 'centrallobjrepresource\_centrallobjrepdest\_policy.json'. Under 'Destination details', 'Source storage account' is listed as 'centrallobjrepresource' and 'Destination storage account' as 'centrallobjrepdest'. In the 'Container pair details' section, it says: 'A container pair consists of a container in the source account and a container in the destination account. Objects in the source container are copied over to the destination container according to the replication rule. You can optionally filter which objects are copied by specifying a prefix match and by copying objects created only after a specified date and time.' A table lists three container pairs:

Source container	Destination container	Filters	Copy over
src-container-1	dest-container-1	1 (view)	Only new objects (view)
src-container-2	dest-container-2	0 (view)	Everything (view)
src-container-3	dest-container-3	0 (view)	Custom (view)

At the bottom, there are 'Upload' and 'Cancel' buttons.

5. Select **Upload** to create the replication policy on the destination account.

You can then download a JSON file containing the policy definition that you can provide to another user to configure the source account. To download this JSON file, follow these steps:

1. Navigate to the **Object replication** settings for the destination account in the Azure portal.
2. Select the **More** button next to the policy that you wish to download, then select **Download rules**, as shown in the following image.

Source account	Source container	Destination container	Filters
centrallobjreplsource	src-container-1	dest-container-1	<ul style="list-style-type: none"><li>Edit rules</li><li>Download rules</li><li>Delete rules</li></ul>
	src-container-2	dest-container-2	<ul style="list-style-type: none"><li>...</li></ul>
	src-container-3	dest-container-3	<ul style="list-style-type: none"><li>...</li></ul>

3. Save the JSON file to your local computer to share with another user to configure the policy on the source account.

The downloaded JSON file includes the policy ID that Azure Storage created for the policy on the destination account. You must use the same policy ID to configure object replication on the source account.

Keep in mind that uploading a JSON file to create a replication policy for the destination account via the Azure portal doesn't automatically create the same policy in the source account. Another user must create the policy on the source account before Azure Storage begins replicating objects.

## Check the replication status of a blob

You can check the replication status for a blob in the source account using the Azure portal, PowerShell, or Azure CLI. Object replication properties aren't populated until replication has either completed or failed.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To check the replication status for a blob in the source account in the Azure portal, follow these steps:

1. Navigate to the source account in the Azure portal.
2. Locate the container that includes the source blob.
3. Select the blob to display its properties. If the blob has been replicated successfully, you'll see in the **Object replication** section that the status is set to *Complete*. The replication policy ID and the ID for the rule governing object replication for this container are also listed.

## blob1.txt

Blob

Save Discard Download Refresh Delete Change tier Acquire lease

Overview Versions Snapshots Edit Generate SAS

### Properties

URL	<a href="https://centralobjreplso...">https://centralobjreplso...</a> 
LAST MODIFIED	10/2/2020, 1:34:16 PM
CREATION TIME	10/2/2020, 1:34:16 PM
VERSION ID	2020-10-02T20:34:16.2859979Z
TYPE	Block blob
SIZE	58 B
ACCESS TIER	Hot (Inferred)
ACCESS TIER LAST MODIFIED	N/A
SERVER ENCRYPTED	true
ETAG	0x8D86712881190BB
CONTENT-TYPE	text/plain
CONTENT-MD5	s76H4wdJ42HQ/ZbxVGY60A==
LEASE STATUS	Unlocked
LEASE STATE	Available
LEASE DURATION	-
COPY STATUS	-
COPY COMPLETION TIME	-

[Undelete](#)

### Object replication

Policy ID	Rule ID	Status
dbe87a1c-6650-479c-8010-af4...	111ff295-d838-47bd-8faf-b09...	Complete

If the replication status for a blob in the source account indicates failure, then investigate the following possible causes:

- Make sure that the object replication policy is configured on the destination account.
- Verify that the destination container still exists.
- If the source blob has been encrypted with a customer-provided key as part of a write operation, then object replication will fail. For more information about customer-provided keys, see [Provide an encryption key on a request to Blob storage](#).

## Remove a replication policy

To remove a replication policy and its associated rules, use Azure portal, PowerShell, or CLI.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To remove a replication policy in the Azure portal, follow these steps:

1. Navigate to the source storage account in the Azure portal.

2. Under **Settings**, select **Object replication**.
3. Select the **More** button next to the policy name.
4. Select **Delete Rules**.

## Next steps

- [Object replication for block blobs](#)
- [Prevent object replication across Azure Active Directory tenants](#)
- [Enable and manage blob versioning](#)
- [Process change feed in Azure Blob storage](#)

# Prevent object replication across Azure Active Directory tenants (preview)

8/22/2022 • 11 minutes to read • [Edit Online](#)

Object replication asynchronously copies block blobs from a container in one storage account to a container in another storage account. When you configure an object replication policy, you specify the source account and container and the destination account and container. After the policy is configured, Azure Storage automatically copies the results of create, update, and delete operations on a source object to the destination object. For more information about object replication in Azure Storage, see [Object replication for block blobs](#).

By default, an authorized user is permitted to configure an object replication policy where the source account is in one Azure Active Directory (Azure AD) tenant, and the destination account is in a different tenant. If your security policies require that you restrict object replication to storage accounts that reside within the same tenant only, you can disallow the creation of policies where the source and destination accounts are in different tenants (preview). By default, cross-tenant object replication is enabled for a storage account unless you explicitly disallow it.

This article describes how to remediate cross-tenant object replication for your storage accounts. It also describes how to create policies to enforce a prohibition on cross-tenant object replication for new and existing storage accounts.

For more information on how to configure object replication policies, including cross-tenant policies, see [Configure object replication for block blobs](#).

## Remediate cross-tenant object replication

To prevent object replication across Azure AD tenants, set the **AllowCrossTenantReplication** property for the storage account to **false**. If a storage account does not currently participate in any cross-tenant object replication policies, then setting the **AllowCrossTenantReplication** property to *false* prevents future configuration of cross-tenant object replication policies with this storage account as the source or destination. However, if a storage account currently participates in one or more cross-tenant object replication policies, then setting the **AllowCrossTenantReplication** property to *false* is not permitted until you delete the existing cross-tenant policies.

Cross-tenant policies are permitted by default for a storage account. However, the **AllowCrossTenantReplication** property is not set by default for a new or existing storage account and does not return a value until you explicitly set it. The storage account can participate in object replication policies across tenants when the property value is either **null** or **true**.

### Remediate cross-tenant replication for a new account

To disallow cross-tenant replication for a new storage account, use the Azure portal, PowerShell, or Azure CLI.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To disallow cross-tenant object replication for a new storage account, follow these steps:

1. In the Azure portal, navigate to the **Storage accounts** page, and select **Create**.
2. Fill out the **Basics** tab for the new storage account.

3. On the **Advanced** tab, in the **Blob storage** section, locate the **Allow cross-tenant replication** setting, and uncheck the box.

The screenshot shows the 'Blob storage' configuration section. It includes an 'Enable network file share v3' checkbox (unchecked), a note about enabling NFS v3, and a 'Allow cross-tenant replication' checkbox (unchecked) which is highlighted with a red border. Below this, there's an 'Access tier' section with two options: 'Hot' (selected) and 'Cool'.

4. Complete the process of creating the account.

#### Remediate cross-tenant replication for an existing account

To disallow cross-tenant replication for an existing storage account, use the Azure portal, PowerShell, or Azure CLI.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To disallow cross-tenant object replication for an existing storage account that is not currently participating in any cross-tenant policies, follow these steps:

1. Navigate to your storage account in the Azure portal.
2. Under **Data management**, select **Object replication**.
3. Select **Advanced settings**.
4. Uncheck **Allow cross-tenant replication**. By default, this box is checked, because cross-tenant object replication is permitted for a storage account unless you explicitly disallow it.

#### Advanced settings

##### Cross-tenant replication

Allowing cross-tenant replication permits you to configure an object replication policy where the source account is in one Azure Active Directory (Azure AD) tenant, and the destination account is in a different tenant.

When you disable this setting for the storage account, all existing object replication policies will be configured to use full Azure Resource Manager resource IDs for both the source and destination account. This enables Azure to determine whether the source and destination accounts are in the same Azure AD tenants.

Allow cross-tenant replication

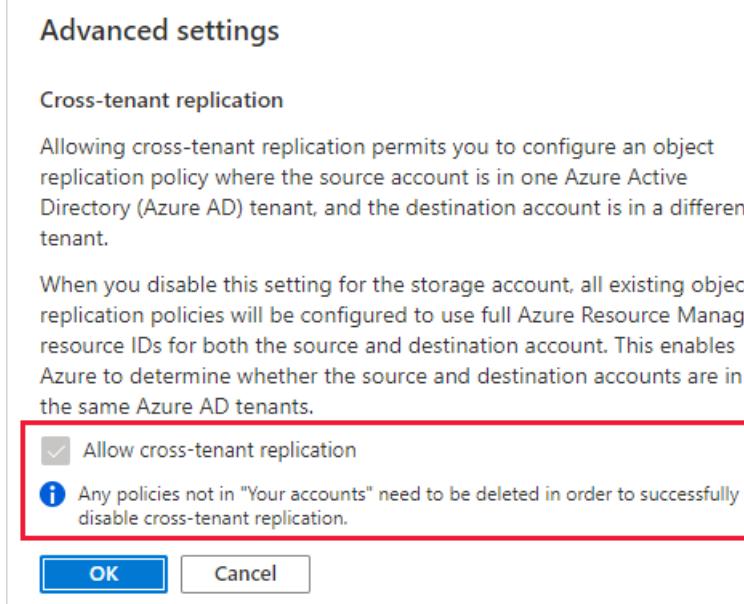
OK

Cancel

5. Select **OK** to save your changes.

If the storage account is currently participating in one or more cross-tenant replication policies, you will not be able to disallow cross-tenant object replication until you delete those policies. In this scenario, the setting is

unavailable in the Azure portal, as shown in the following image.



After you disallow cross-tenant replication, attempting to configure a cross-tenant policy with the storage account as the source or destination fails. Azure Storage returns an error indicating that cross-tenant object replication is not permitted for the storage account.

When cross-tenant object replication is disallowed for a storage account, then any new object replication policies that you create with that account must include the full Azure Resource Manager IDs for the source and destination account. Azure Storage requires the full resource ID to verify whether the source and destination accounts reside within the same tenant. For more information, see [Specify full resource IDs for the source and destination accounts](#).

The **AllowCrossTenantReplication** property is supported for storage accounts that use the Azure Resource Manager deployment model only. For information about which storage accounts use the Azure Resource Manager deployment model, see [Types of storage accounts](#).

## Permissions for allowing or disallowing cross-tenant replication

To set the **AllowCrossTenantReplication** property for a storage account, a user must have permissions to create and manage storage accounts. Azure role-based access control (Azure RBAC) roles that provide these permissions include the **Microsoft.Storage/storageAccounts/write** or **Microsoft.Storage/storageAccounts/\*** action. Built-in roles with this action include:

- The Azure Resource Manager [Owner](#) role
- The Azure Resource Manager [Contributor](#) role
- The [Storage Account Contributor](#) role

These roles do not provide access to data in a storage account via Azure Active Directory (Azure AD). However, they include the **Microsoft.Storage/storageAccounts/listkeys/action**, which grants access to the account access keys. With this permission, a user can use the account access keys to access all data in a storage account.

Role assignments must be scoped to the level of the storage account or higher to permit a user to allow or disallow cross-tenant object replication for the storage account. For more information about role scope, see [Understand scope for Azure RBAC](#).

Be careful to restrict assignment of these roles only to those who require the ability to create a storage account or update its properties. Use the principle of least privilege to ensure that users have the fewest permissions that they need to accomplish their tasks. For more information about managing access with Azure RBAC, see [Best practices for Azure RBAC](#).

#### NOTE

The classic subscription administrator roles Service Administrator and Co-Administrator include the equivalent of the Azure Resource Manager [Owner](#) role. The [Owner](#) role includes all actions, so a user with one of these administrative roles can also create and manage storage accounts. For more information, see [Classic subscription administrator roles, Azure roles, and Azure AD administrator roles](#).

## Use Azure Policy to audit for compliance

If you have a large number of storage accounts, you may want to perform an audit to make sure that those accounts are configured to prevent cross-tenant object replication. To audit a set of storage accounts for their compliance, use Azure Policy. Azure Policy is a service that you can use to create, assign, and manage policies that apply rules to Azure resources. Azure Policy helps you to keep those resources compliant with your corporate standards and service level agreements. For more information, see [Overview of Azure Policy](#).

### Create a policy with an Audit effect

Azure Policy supports effects that determine what happens when a policy rule is evaluated against a resource. The Audit effect creates a warning when a resource is not in compliance, but does not stop the request. For more information about effects, see [Understand Azure Policy effects](#).

To create a policy with an Audit effect for the cross-tenant object replication setting for a storage account with the Azure portal, follow these steps:

1. In the Azure portal, navigate to the Azure Policy service.
2. Under the **Authoring** section, select **Definitions**.
3. Select **Add policy definition** to create a new policy definition.
4. For the **Definition location** field, select the **More** button to specify where the audit policy resource is located.
5. Specify a name for the policy. You can optionally specify a description and category.
6. Under **Policy rule**, add the following policy definition to the **policyRule** section.

```
{
 "if": {
 "allOf": [
 {
 "field": "type",
 "equals": "Microsoft.Storage/storageAccounts"
 },
 {
 "not": {
 "field": "Microsoft.Storage/storageAccounts/allowCrossTenantReplication",
 "equals": "false"
 }
 }
]
 },
 "then": {
 "effect": "audit"
 }
}
```

7. Save the policy.

### Assign the policy

Next, assign the policy to a resource. The scope of the policy corresponds to that resource and any resources beneath it. For more information on policy assignment, see [Azure Policy assignment structure](#).

To assign the policy with the Azure portal, follow these steps:

1. In the Azure portal, navigate to the Azure Policy service.
2. Under the **Authoring** section, select **Assignments**.
3. Select **Assign policy** to create a new policy assignment.
4. For the **Scope** field, select the scope of the policy assignment.
5. For the **Policy definition** field, select the **More** button, then select the policy you defined in the previous section from the list.
6. Provide a name for the policy assignment. The description is optional.
7. Leave **Policy enforcement** set to *Enabled*. This setting has no effect on the audit policy.
8. Select **Review + create** to create the assignment.

### **View compliance report**

After you've assigned the policy, you can view the compliance report. The compliance report for an audit policy provides information on which storage accounts are still permitting cross-tenant object replication policies. For more information, see [Get policy compliance data](#).

It may take several minutes for the compliance report to become available after the policy assignment is created.

To view the compliance report in the Azure portal, follow these steps:

1. In the Azure portal, navigate to the Azure Policy service.
2. Select **Compliance**.
3. Filter the results for the name of the policy assignment that you created in the previous step. The report shows resources that are not in compliance with the policy.
4. You can drill down into the report for additional details, including a list of storage accounts that are not in compliance.

## audit-policy-cross-tenant-replication

Policy compliance

X

[View definition](#) [Edit assignment](#) [Assign to another scope](#) [Delete assignment](#) [Create Remediation Task](#) [Create exemption](#)

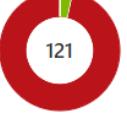
## ^ Essentials

Name audit-policy-cross-tenant-replication	Scope Azure Storage content development and testing/storagesamples-rg
Description --	Excluded scopes 0
Assignment ID <assignment-id>	Definition audit-policy-cross-tenant-replication

**Selected Scopes** ⓘ  
2 selected subscriptions ⏺

**Compliance state** ⓘ  
 Non-compliant

**Overall resource compliance** ⓘ  
3%  
 4 out of 121

**Resources by compliance state** ⓘ  


4 - Compliant
0 - Exempt
117 - Non-compliant

**Details**  
**Effect Type** Audit  
**Parent Initiative** <<NONE>>

**Resource compliance** Events

Filter by resource name or ID...		Non-compliant	microsoft.sto...	All locations	
Name	Compliance state	Compliance reas...	Resource Type	Location	Scope
allowsharedkeyaccess	Non-compliant	Details	microsoft.storage/st...	West US 2	Azure Storage content
anonympublicaccess	Non-compliant	Details	microsoft.storage/st...	East US	Azure Storage content
anonymousaccesslogs	Non-compliant	Details	microsoft.storage/st...	East US	Azure Storage content

## Use Azure Policy to enforce same-tenant replication policies

Azure Policy supports cloud governance by ensuring that Azure resources adhere to requirements and standards. To ensure that storage accounts in your organization disallow cross-tenant replication, you can create a policy that prevents the creation of a new storage account that allows cross-tenant object replication policies. The enforcement policy uses the Deny effect to prevent a request that would create or modify a storage account to allow cross-tenant object replication. The Deny policy will also prevent all configuration changes to an existing account if the cross-tenant object replication setting for that account is not compliant with the policy. For more information about the Deny effect, see [Understand Azure Policy effects](#).

To create a policy with a Deny effect for cross-tenant object replication, follow the same steps described in [Use Azure Policy to audit for compliance](#), but provide the following JSON in the `policyRule` section of the policy definition:

```
{
 "if": {
 "allOf": [
 {
 "field": "type",
 "equals": "Microsoft.Storage/storageAccounts"
 },
 {
 "not": {
 "field": "Microsoft.Storage/storageAccounts/allowCrossTenantReplication",
 "equals": "false"
 }
 }
]
 },
 "then": {
 "effect": "deny"
 }
}
```

After you create the policy with the Deny effect and assign it to a scope, a user cannot create a storage account that allows cross-tenant object replication. Nor can a user make any configuration changes to an existing storage account that currently allows cross-tenant object replication. Attempting to do so results in an error. The **AllowCrossTenantReplication** property for the storage account must be set to **false** to proceed with account creation or configuration updates, in compliance with the policy.

The following image shows the error that occurs if you try to create a storage account that allows cross-tenant object replication (the default for a new account) when a policy with a Deny effect requires that cross-tenant object replication is disallowed.

## Errors

[Summary](#) [Raw Error](#)

ERROR DETAILS 

Resource 'allowcrosstenant' was disallowed by policy. (Code: RequestDisallowedByPolicy)

Policy: [deny-policy-cross-tenant-replication](#)

WAS THIS HELPFUL?  

Troubleshooting Options

[Check Usage + Quota](#)   
[New Support Request](#) 

## See also

- [Object replication for block blobs](#)
- [Configure object replication for block blobs](#)

# Managing Concurrency in Blob storage

8/22/2022 • 9 minutes to read • [Edit Online](#)

Modern applications often have multiple users viewing and updating data simultaneously. Application developers need to think carefully about how to provide a predictable experience to their end users, particularly for scenarios where multiple users can update the same data. There are three main data concurrency strategies that developers typically consider:

- **Optimistic concurrency:** An application performing an update will, as part of its update, determine whether the data has changed since the application last read that data. For example, if two users viewing a wiki page make an update to that page, then the wiki platform must ensure that the second update does not overwrite the first update. It must also ensure that both users understand whether their update was successful. This strategy is most often used in web applications.
- **Pessimistic concurrency:** An application looking to perform an update will take a lock on an object preventing other users from updating the data until the lock is released. For example, in a primary/secondary data replication scenario in which only the primary performs updates, the primary typically holds an exclusive lock on the data for an extended period of time to ensure no one else can update it.
- **Last writer wins:** An approach that allows update operations to proceed without first determining whether another application has updated the data since it was read. This approach is typically used when data is partitioned in such a way that multiple users will not access the same data at the same time. It can also be useful where short-lived data streams are being processed.

Azure Storage supports all three strategies, although it is distinctive in its ability to provide full support for optimistic and pessimistic concurrency. Azure Storage was designed to embrace a strong consistency model that guarantees that after the service performs an insert or update operation, subsequent read operations return the latest update.

In addition to selecting an appropriate concurrency strategy, developers should also be aware of how a storage platform isolates changes, particularly changes to the same object across transactions. Azure Storage uses snapshot isolation to allow read operations concurrently with write operations within a single partition. Snapshot isolation guarantees that all read operations return a consistent snapshot of the data even while updates are occurring.

You can opt to use either optimistic or pessimistic concurrency models to manage access to blobs and containers. If you don't explicitly specify a strategy, then by default the last writer wins.

## Optimistic concurrency

Azure Storage assigns an identifier to every object stored. This identifier is updated every time a write operation is performed on an object. The identifier is returned to the client as part of an HTTP GET response in the ETag header that is defined by the HTTP protocol.

A client that is performing an update can send the original ETag together with a conditional header to ensure that an update will only occur if a certain condition has been met. For example, if the **If-Match** header is specified, Azure Storage verifies that the value of the ETag specified in the update request is the same as the ETag for the object being updated. For more information about conditional headers, see [Specifying conditional headers for Blob service operations](#).

The outline of this process is as follows:

1. Retrieve a blob from Azure Storage. The response includes an HTTP ETag Header value that identifies the current version of the object.
2. When you update the blob, include the ETag value you received in step 1 in the **If-Match** conditional header of the write request. Azure Storage compares the ETag value in the request with the current ETag value of the blob.
3. If the blob's current ETag value differs from that specified in the **If-Match** conditional header provided on the request, then Azure Storage returns HTTP status code 412 (Precondition Failed). This error indicates to the client that another process has updated the blob since the client first retrieved it.
4. If the current ETag value of the blob is the same version as the ETag in the **If-Match** conditional header in the request, Azure Storage performs the requested operation and updates the current ETag value of the blob.

The following code examples show how to construct an **If-Match** condition on the write request that checks the ETag value for a blob. Azure Storage evaluates whether the blob's current ETag is the same as the ETag provided on the request and performs the write operation only if the two ETag values match. If another process has updated the blob in the interim, then Azure Storage returns an HTTP 412 (Precondition Failed) status message.

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

```
private static async Task DemonstrateOptimisticConcurrencyBlob(BlobClient blobClient)
{
 Console.WriteLine("Demonstrate optimistic concurrency");

 BlobContainerClient containerClient = blobClient.GetParentBlobContainerClient();

 try
 {
 // Create the container if it does not exist.
 await containerClient.CreateIfNotExistsAsync();

 // Upload text to a new block blob.
 string blobContents1 = "First update. Overwrite blob if it exists.";
 byte[] byteArray = Encoding.ASCII.GetBytes(blobContents1);

 ETag originalETag;

 using (MemoryStream stream = new MemoryStream(byteArray))
 {
 BlobContentInfo blobContentInfo = await blobClient.UploadAsync(stream, overwrite: true);
 originalETag = blobContentInfo.ETag;
 Console.WriteLine("Blob added. Original ETag = {0}", originalETag);
 }

 // This code simulates an update by another client.
 // No ETag was provided, so original blob is overwritten and ETag updated.
 string blobContents2 = "Second update overwrites first update.";
 byteArray = Encoding.ASCII.GetBytes(blobContents2);

 using (MemoryStream stream = new MemoryStream(byteArray))
 {
 BlobContentInfo blobContentInfo = await blobClient.UploadAsync(stream, overwrite: true);
 Console.WriteLine("Blob updated. Updated ETag = {0}", blobContentInfo.ETag);
 }

 // Now try to update the blob using the original ETag value.
 string blobContents3 = "Third update. If-Match condition set to original ETag.";
 byteArray = Encoding.ASCII.GetBytes(blobContents3);

 // Set the If-Match condition to the original ETag.
 BlobUploadOptions blobUploadOptions = new BlobUploadOptions()
 {
 Conditions = new BlobRequestConditions()
 }
 }
}
```

```

 {
 IfMatch = originalETag
 }
 };

 using (MemoryStream stream = new MemoryStream(byteArray))
 {
 // This call should fail with error code 412 (Precondition Failed).
 BlobContentInfo blobContentInfo = await blobClient.UploadAsync(stream, blobUploadOptions);
 }
}
catch (RequestFailedException e)
{
 if (e.Status == (int)HttpStatusCode.PreconditionFailed)
 {
 Console.WriteLine(
 @"Precondition failure as expected. Blob's ETag does not match ETag provided.");
 }
 else
 {
 Console.WriteLine(e.Message);
 throw;
 }
}
}
}

```

Azure Storage also supports other conditional headers, including as **If-Modified-Since**, **If-Unmodified-Since** and **If-None-Match**. For more information, see [Specifying Conditional Headers for Blob Service Operations](#).

## Pessimistic concurrency for blobs

To lock a blob for exclusive use, you can acquire a lease on it. When you acquire the lease, you specify the duration of the lease. A finite lease may be valid from between 15 to 60 seconds. A lease can also be infinite, which amounts to an exclusive lock. You can renew a finite lease to extend it, and you can release the lease when you're finished with it. Azure Storage automatically releases finite leases when they expire.

Leases enable different synchronization strategies to be supported, including exclusive write/shared read operations, exclusive write/exclusive read operations, and shared write/exclusive read operations. When a lease exists, Azure Storage enforces exclusive access to write operations for the lease holder. However, ensuring exclusivity for read operations requires the developer to make sure that all client applications use a lease ID and that only one client at a time has a valid lease ID. Read operations that do not include a lease ID result in shared reads.

The following code examples show how to acquire an exclusive lease on a blob, update the content of the blob by providing the lease ID, and then release the lease. If the lease is active and the lease ID is not provided on a write request, then the write operation fails with error code 412 (Precondition Failed).

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

```

public static async Task DemonstratePessimisticConcurrencyBlob(BlobClient blobClient)
{
 Console.WriteLine("Demonstrate pessimistic concurrency");

 BlobContainerClient containerClient = blobClient.GetParentBlobContainerClient();
 BlobLeaseClient blobLeaseClient = blobClient.GetBlobLeaseClient();

 try
 {
 // Create the container if it does not exist.
 await containerClient.CreateIfNotExistsAsync();
 }
}

```

```

// Upload text to a blob.
string blobContents1 = "First update. Overwrite blob if it exists.";
byte[] byteArray = Encoding.ASCII.GetBytes(blobContents1);
using (MemoryStream stream = new MemoryStream(byteArray))
{
 BlobContentInfo blobContentInfo = await blobClient.UploadAsync(stream, overwrite: true);
}

// Acquire a lease on the blob.
BlobLease blobLease = await blobLeaseClient.AcquireAsync(TimeSpan.FromSeconds(15));
Console.WriteLine("Blob lease acquired. LeaseId = {0}", blobLease.LeaseId);

// Set the request condition to include the lease ID.
BlobUploadOptions blobUploadOptions = new BlobUploadOptions()
{
 Conditions = new BlobRequestConditions()
 {
 LeaseId = blobLease.LeaseId
 }
};

// Write to the blob again, providing the lease ID on the request.
// The lease ID was provided, so this call should succeed.
string blobContents2 = "Second update. Lease ID provided on request.";
byteArray = Encoding.ASCII.GetBytes(blobContents2);

using (MemoryStream stream = new MemoryStream(byteArray))
{
 BlobContentInfo blobContentInfo = await blobClient.UploadAsync(stream, blobUploadOptions);
}

// This code simulates an update by another client.
// The lease ID is not provided, so this call fails.
string blobContents3 = "Third update. No lease ID provided.";
byteArray = Encoding.ASCII.GetBytes(blobContents3);

using (MemoryStream stream = new MemoryStream(byteArray))
{
 // This call should fail with error code 412 (Precondition Failed).
 BlobContentInfo blobContentInfo = await blobClient.UploadAsync(stream);
}
}

catch (RequestFailedException e)
{
 if (e.Status == (int) HttpStatusCode.PreconditionFailed)
 {
 Console.WriteLine(
 @"Precondition failure as expected. The lease ID was not provided.");
 }
 else
 {
 Console.WriteLine(e.Message);
 throw;
 }
}
finally
{
 await blobLeaseClient.ReleaseAsync();
}
}

```

## Pessimistic concurrency for containers

Leases on containers enable the same synchronization strategies that are supported for blobs, including exclusive write/shared read, exclusive write/exclusive read, and shared write/exclusive read. For containers,

however, the exclusive lock is enforced only on delete operations. To delete a container with an active lease, a client must include the active lease ID with the delete request. All other container operations will succeed on a leased container without the lease ID.

## Next steps

- [Specifying conditional headers for Blob service operations](#)
- [Lease Container](#)
- [Lease Blob](#)

# Host a static website in Azure Storage

8/22/2022 • 7 minutes to read • [Edit Online](#)

You can serve static content (HTML, CSS, JavaScript, and image files) directly from a container in a [general-purpose V2](#) or [BlockBlobStorage](#) account. To learn more, see [Static website hosting in Azure Storage](#).

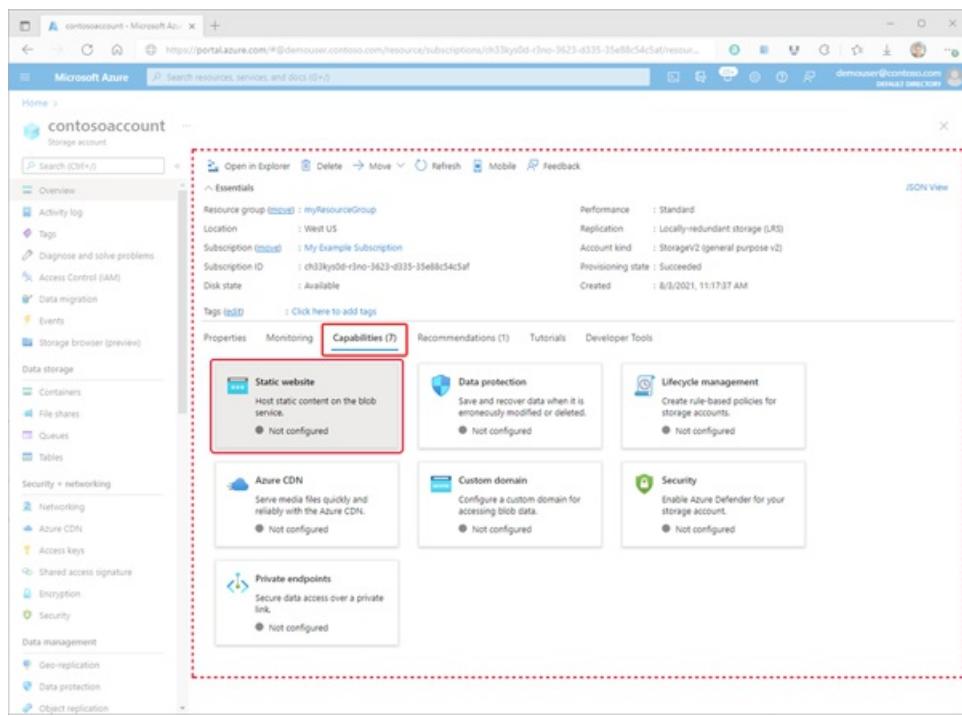
This article shows you how to enable static website hosting by using the Azure portal, the Azure CLI, or PowerShell.

## Enable static website hosting

Static website hosting is a feature that you have to enable on the storage account.

- [Portal](#)
- [Azure CLI](#)
- [PowerShell](#)

1. Sign-in to the [Azure portal](#) to get started.
2. Locate your storage account and select it to display the account's **Overview** pane.
3. In the **Overview** pane, select the **Capabilities** tab. Next, select **Static website** to display the configuration page for the static website.



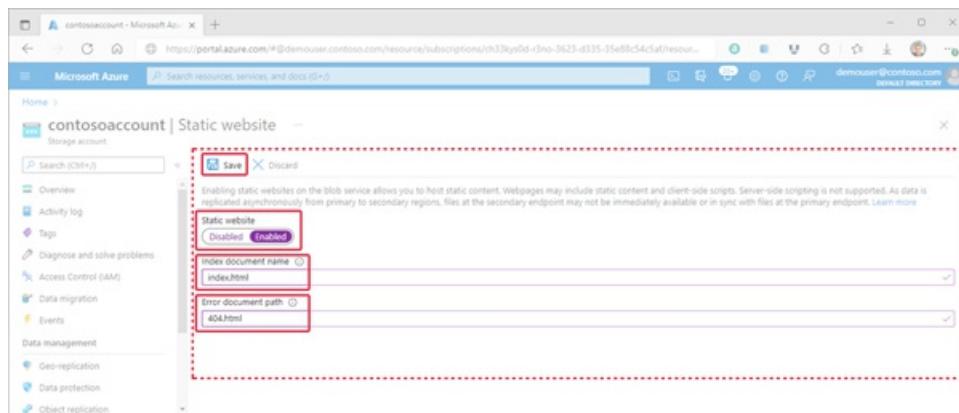
4. Select **Enabled** to enable static website hosting for the storage account.
5. In the **Index document name** field, specify a default index page (For example: *index.html*).

The default index page is displayed when a user navigates to the root of your static website.

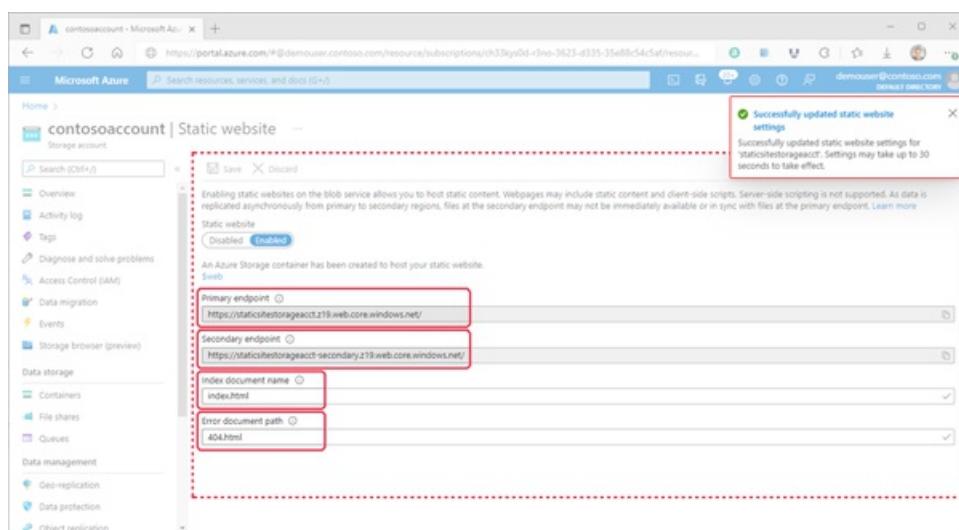
6. In the **Error document path** field, specify a default error page (For example: *404.html*).

The default error page is displayed when a user attempts to navigate to a page that does not exist in your static website.

7. Click **Save** to finish the static site configuration.



8. A confirmation message is displayed. Your static website endpoints and other configuration information are shown within the **Overview** pane.



## Upload files

- [Portal](#)
- [Azure CLI](#)
- [PowerShell](#)

The following instructions show you how to upload files by using the Azure portal. You could also use [AzCopy](#), PowerShell, CLI, or any custom application that can upload files to the `$web` container of your account. For a step-by-step tutorial that uploads files by using Visual Studio code, see [Tutorial: Host a static website on Blob Storage](#).

1. In the Azure portal, navigate to the storage account containing your static website. Select **Containers** in the left navigation pane to display the list of containers.
2. In the **Containers** pane, select the `$web` container to open the container's **Overview** pane.

The screenshot shows the Microsoft Azure Storage account overview page for 'contosoaccount'. In the left sidebar, under 'Data storage', the 'Containers' link is highlighted with a red box. The main pane displays a table of containers, with one row for '\$web' selected.

3. In the **Overview** pane, select the **Upload** icon to open the **Upload blob** pane. Next, select the **Files** field within the **Upload blob** pane to open the file browser. Navigate to the file you want to upload, select it, and then select **Open** to populate the **Files** field. Optionally, select the **Overwrite if files already exist** checkbox.

The screenshot shows the 'Upload blob' pane for the '\$web' container. The 'Upload' icon in the top bar is highlighted with a red box. The 'Files' input field, which contains 'MyNewWebpage.htm', is also highlighted with a red box. The 'Upload' button at the bottom of the pane is also highlighted with a red box.

4. If you intend for the browser to display the contents of file, make sure that the content type of that file is set to `text/html`. To verify this, select the name of the blob you uploaded in the previous step to open its **Overview** pane. Ensure that the value is set within the **CONTENT-TYPE** property field.

The screenshot shows the 'Overview' pane for the blob 'MyNewWebpage.htm'. The 'CONTENT-TYPE' field is highlighted with a red box and contains the value 'text/html'.

#### NOTE

This property is automatically set to `text/html` for commonly recognized extensions such as `.html`. However, in some cases, you'll have to set this yourself. If you don't set this property to `text/html`, the browser will prompt users to download the file instead of rendering the contents. This property can be set in the previous step.

## Find the website URL

You can view the pages of your site from a browser by using the public URL of the website.

- [Portal](#)
- [Azure CLI](#)
- [PowerShell](#)

In the pane that appears beside the account overview page of your storage account, select **Static Website**. The URL of your site appears in the **Primary endpoint** field.

The screenshot shows the Azure Storage account overview for 'contoso'. On the left, there's a sidebar with various navigation options like Overview, Activity log, Tags, etc. The 'Static website' section is currently selected. It displays a status message about enabling static websites on the blob service. Below that, it shows the 'Static website' status as 'Enabled' (with a blue button) and a note that an Azure Storage container has been created to host the static website. The 'Primary endpoint' field contains the URL <https://contoso.z22.web.core.windows.net/>, which is highlighted with a red box.

## Enable metrics on static website pages

Once you've enabled metrics, traffic statistics on files in the **\$web** container are reported in the metrics dashboard.

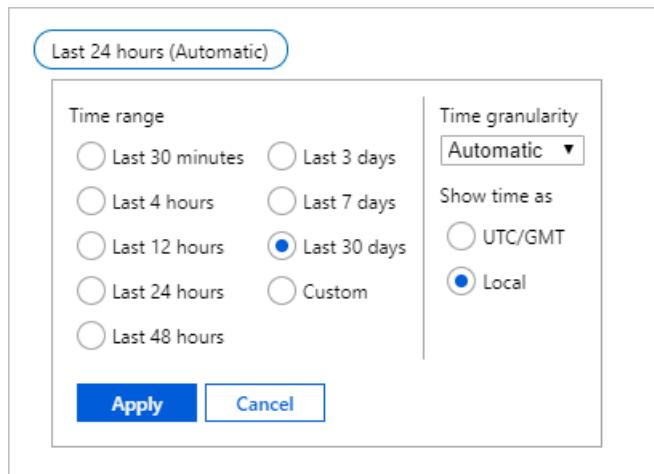
1. Click **Metrics** under the **Monitor** section of the storage account menu.

The screenshot shows the 'Monitoring' section of the storage account menu. It includes links for Insights, Alerts, Metrics (which is highlighted with a red box), and Workbooks.

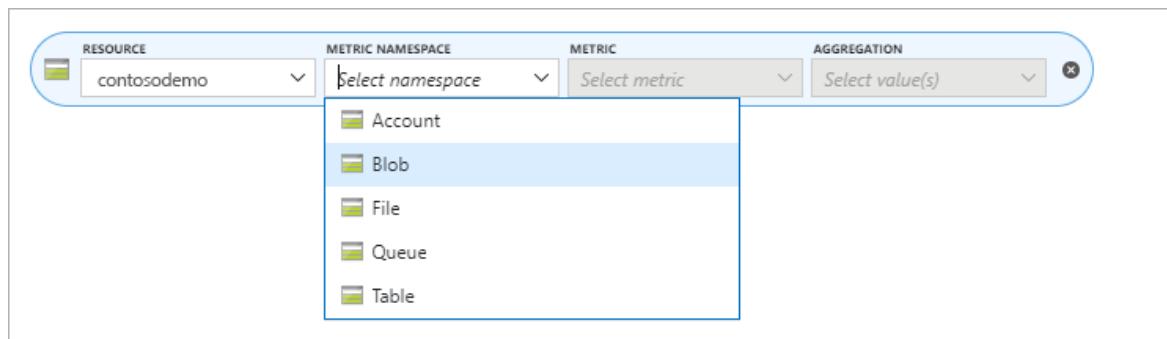
### NOTE

Metrics data are generated by hooking into different metrics APIs. The portal only displays API members used within a given time frame in order to only focus on members that return data. In order to ensure you're able to select the necessary API member, the first step is to expand the time frame.

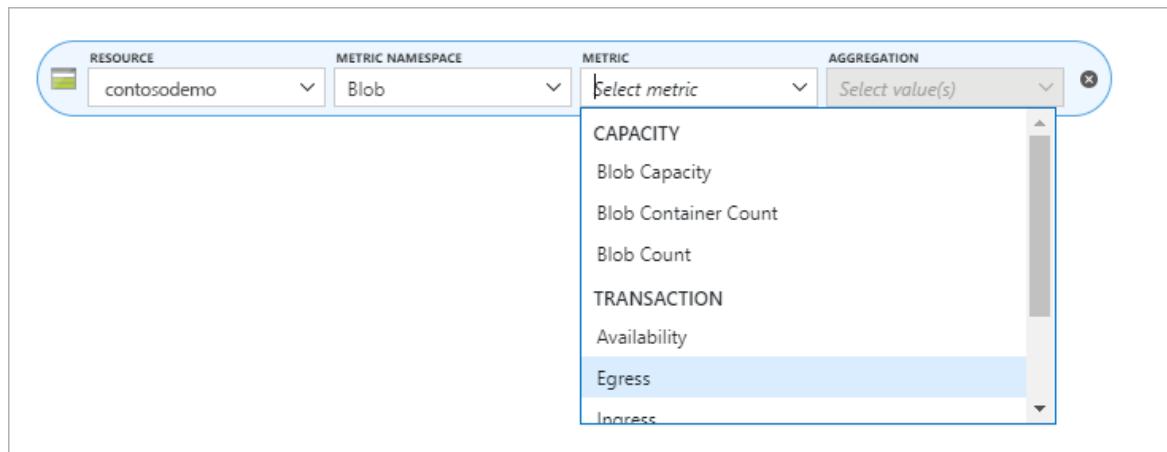
2. Click on the time frame button, choose a time frame, and then click **Apply**.



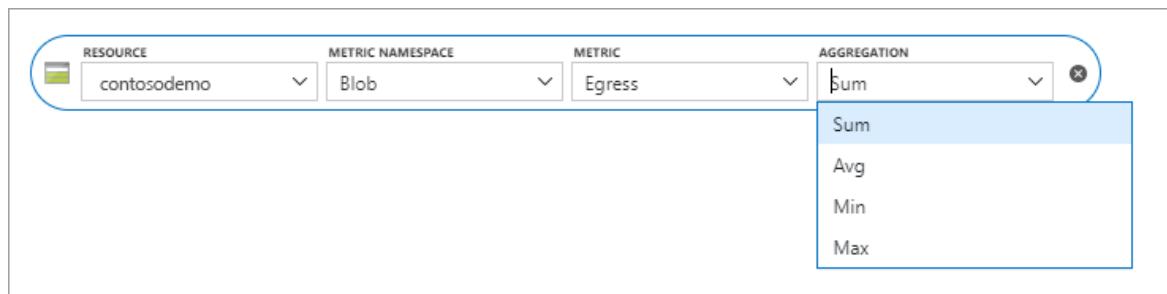
3. Select **Blob** from the *Namespace* drop down.



4. Then select the **Egress** metric.



5. Select **Sum** from the *Aggregation* selector.



6. Click the **Add filter** button and choose **API name** from the *Property* selector.

The screenshot shows the Azure Metrics Explorer interface. At the top, there's a search bar with the text "contosodemo, Egress, Sum" and a "Add metric" button. Below the search bar, there are two dropdown menus: "PROPERTY" (set to "Select property") and "VALUES" (set to "Select value(s)"). A list of properties is displayed in a scrollable box: "Geo type", "API name", and "Authentication".

7. Check the box next to **GetWebContent** in the *Values* selector to populate the metrics report.

The screenshot shows the Azure Metrics Explorer interface. The "PROPERTY" dropdown is set to "API name" and the "VALUES" dropdown is set to "GetWebContent". A scrollable list of API members is shown, with "GetWebContent" checked. Other members listed include GetBlobServiceProperties, BlobPreflightRequest, SetBlobServiceProperties, GetContainerProperties, DeleteBlob, GetBlobProperties, and GetContainerACL.

#### NOTE

The **GetWebContent** checkbox appears only if that API member was used within a given time frame. The portal only displays API members used within a given time frame in order to only focus on members that return data. If you can't find a specific API member in this list, expand the time frame.

## Next steps

- Learn how to configure a custom domain with your static website. See [Map a custom domain to an Azure Blob Storage endpoint](#).

# Integrate a static website with Azure CDN

8/22/2022 • 3 minutes to read • [Edit Online](#)

You can enable [Azure Content Delivery Network \(CDN\)](#) to cache content from a [static website](#) that is hosted in an Azure storage account. You can use Azure CDN to configure the custom domain endpoint for your static website, provision custom TLS/SSL certificates, and configure custom rewrite rules. Configuring Azure CDN results in additional charges, but provides consistent low latencies to your website from anywhere in the world. Azure CDN also provides TLS encryption with your own certificate.

For information on Azure CDN pricing, see [Azure CDN pricing](#).

## Enable Azure CDN for your static website

You can enable Azure CDN for your static website directly from your storage account. If you want to specify advanced configuration settings for your CDN endpoint, such as [large file download optimization](#), you can instead use the [Azure CDN extension](#) to create a CDN profile and endpoint.

1. Locate your storage account in the Azure portal and display the account overview.
2. Under the **Security + Networking** menu, select **Azure CDN** to open the **Azure CDN** page:

The screenshot shows the 'New endpoint' configuration page. It includes fields for 'CDN profile' (radio buttons for 'Create new' or 'Use existing'), 'Pricing tier' (dropdown menu), 'CDN endpoint name' (text input field ending in '.azureedge.net'), and 'Origin hostname' (text input field containing 'storagesamplesstatic.z5.web.core.windows.net'). A large blue 'Create' button is at the bottom.

3. In the **CDN profile** section, specify whether to create a new CDN profile or use an existing one. A CDN profile is a collection of CDN endpoints that share a pricing tier and provider. Then enter a name for the CDN that's unique within your subscription.
4. Specify a pricing tier for the CDN endpoint. To learn more about pricing, see [Content Delivery Network pricing](#). For more information about the features available with each tier, see [Compare Azure CDN product features](#).
5. In the **CDN endpoint name** field, specify a name for your CDN endpoint. The CDN endpoint must be unique across Azure and provides the first part of the endpoint URL. The form validates that the endpoint name is unique.
6. Specify your static website endpoint in the **Origin hostname** field.

To find your static website endpoint, navigate to the **Static website** settings for your storage account.

Copy the primary endpoint and paste it into the CDN configuration.

#### IMPORTANT

Make sure to remove the protocol identifier (*e.g.*, HTTPS) and the trailing slash in the URL. For example, if the static website endpoint is `https://mystorageaccount.z5.web.core.windows.net/`, then you would specify `mystorageaccount.z5.web.core.windows.net` in the **Origin hostname** field.

The following image shows an example endpoint configuration:

New endpoint

CDN profile ⓘ  
 Create new  Use existing

static-website-samples ✓

Pricing tier (View full pricing details) \*  
Standard Akamai

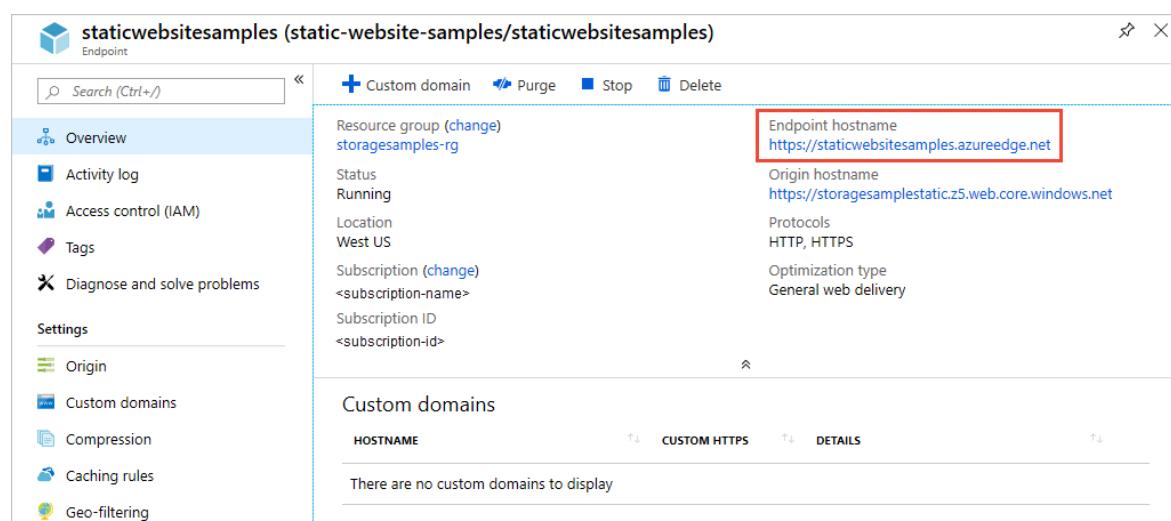
CDN endpoint name \*  
staticwebsitesamples ✓  
.azureedge.net

Origin hostname ⓘ  
storagesamplestatic.z5.web.core.windows.net

**Create**

7. Select **Create**, and then wait for the CDN to provision. After the endpoint is created, it appears in the endpoint list. (If you have any errors in the form, an exclamation mark appears next to that field.)

8. To verify that the CDN endpoint is configured correctly, click on the endpoint to navigate to its settings. From the CDN overview for your storage account, locate the endpoint hostname, and navigate to the endpoint, as shown in the following image. The format of your CDN endpoint will be similar to `https://staticwebsitesamples.azureedge.net`.



staticwebsitesamples (static-website-samples/staticwebsitesamples)  
Endpoint

Custom domain Purge Stop Delete

Overview

Resource group (change)  
storagesamples-rg

Status  
Running

Location  
West US

Subscription (change)  
<subscription-name>

Subscription ID  
<subscription-id>

Endpoint hostname  
`https://staticwebsitesamples.azureedge.net`

Origin hostname  
`https://storagesamplestatic.z5.web.core.windows.net`

Protocols  
HTTP, HTTPS

Optimization type  
General web delivery

Custom domains

HOSTNAME	CUSTOM HTTPS	DETAILS
There are no custom domains to display		

9. Once the CDN endpoint is provisioned, navigating to the CDN endpoint displays the contents of the index.html file that you previously uploaded to your static website.
10. To review the origin settings for your CDN endpoint, navigate to **Origin** under the **Settings** section for your CDN endpoint. You will see that the **Origin type** field is set to *Custom Origin* and that the **Origin hostname** field displays your static website endpoint.

The screenshot shows the Azure portal interface for managing a static website sample endpoint named 'staticwebsitesamples'. The left sidebar lists navigation options: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (with 'Origin' selected), Custom domains, Compression, Caching rules, and Geo-filtering. The main content area is titled 'Origin' and contains fields for configuration: 'Origin type' (set to 'Custom origin'), 'Origin hostname' (set to 'storagesamplestatic.z5.web.core.windows.net'), 'Origin host header' (set to 'storagesamplestatic.z5.web.core.windows.net'), 'Origin path' (set to '/Path'), 'Protocol' (checkboxes for 'HTTP' checked and 'HTTPS' checked), 'Origin port' (set to '80' for HTTP and '443' for HTTPS). Buttons for 'Save' and 'Discard' are at the top right.

## Remove content from Azure CDN

If you no longer want to cache an object in Azure CDN, you can take one of the following steps:

- Make the container private instead of public. For more information, see [Manage anonymous read access to containers and blobs](#).
- Disable or delete the CDN endpoint by using the Azure portal.
- Modify your hosted service to no longer respond to requests for the object.

An object that's already cached in Azure CDN remains cached until the time-to-live period for the object expires or until the endpoint is [purged](#). When the time-to-live period expires, Azure CDN determines whether the CDN endpoint is still valid and the object is still anonymously accessible. If they are not, the object will no longer be cached.

## Next steps

(Optional) Add a custom domain to your Azure CDN endpoint. See [Tutorial: Add a custom domain to your Azure CDN endpoint](#).

# Use GitHub Actions workflow to deploy your static website in Azure Storage

8/22/2022 • 8 minutes to read • [Edit Online](#)

Get started with [GitHub Actions](#) by using a workflow to deploy a static site to an Azure storage account. Once you have set up a GitHub Actions workflow, you will be able to automatically deploy your site to Azure from GitHub when you make changes to your site's code.

## NOTE

If you are using [Azure Static Web Apps](#), then you do not need to manually set up a GitHub Actions workflow. Azure Static Web Apps automatically creates a GitHub Actions workflow for you.

## Prerequisites

An Azure subscription and GitHub account.

- An Azure account with an active subscription. [Create an account for free](#).
- A GitHub repository with your static website code. If you do not have a GitHub account, [sign up for free](#).
- A working static website hosted in Azure Storage. Learn how to [host a static website in Azure Storage](#). To follow this example, you should also deploy [Azure CDN](#).

## NOTE

It's common to use a content delivery network (CDN) to reduce latency to your users around the globe and to reduce the number of transactions to your storage account. Deploying static content to a cloud-based storage service can reduce the need for potentially expensive compute instance. For more information, see [Static Content Hosting pattern](#).

## Generate deployment credentials

- [Service principal](#)
- [OpenID Connect](#)

You can create a [service principal](#) with the `az ad sp create-for-rbac` command in the [Azure CLI](#). Run this command with [Azure Cloud Shell](#) in the Azure portal or by selecting the [Try it](#) button.

Replace the placeholder `myStaticSite` with the name of your site hosted in Azure Storage.

```
az ad sp create-for-rbac --name {myStaticSite} --role contributor --scopes /subscriptions/{subscription-id}/resourceGroups/{resource-group} --sdk-auth
```

In the example, replace the placeholders with your subscription ID and resource group name. The output is a JSON object with the role assignment credentials that provide access to your storage account. Copy this JSON object for later.

```
{
 "clientId": "<GUID>",
 "clientSecret": "<GUID>",
 "subscriptionId": "<GUID>",
 "tenantId": "<GUID>",
 (...),
}
```

#### IMPORTANT

It is always a good practice to grant minimum access. The scope in the previous example is limited to the specific App Service app and not the entire resource group.

## Configure the GitHub secret

- [Service principal](#)
- [OpenID Connect](#)

1. In [GitHub](#), browse your repository.
2. Select **Settings > Secrets > New secret**.
3. Paste the entire JSON output from the Azure CLI command into the secret's value field. Give the secret a name like `AZURE_CREDENTIALS`.

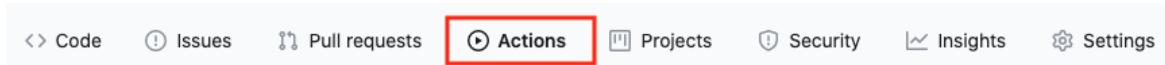
When you configure the workflow file later, you use the secret for the input `creds` of the Azure Login action. For example:

```
- uses: azure/login@v1
 with:
 creds: ${{ secrets.AZURE_CREDENTIALS }}
```

## Add your workflow

- [Service principal](#)
- [OpenID Connect](#)

1. Go to **Actions** for your GitHub repository.



2. Select **Set up your workflow yourself**.
3. Delete everything after the `on:` section of your workflow file. For example, your remaining workflow may look like this.

```
name: CI

on:
 push:
 branches: [main]
```

4. Rename your workflow `Blob storage website CI` and add the checkout and login actions. These actions

will check out your site code and authenticate with Azure using the `AZURE_CREDENTIALS` GitHub secret you created earlier.

```
name: Blob storage website CI

on:
 push:
 branches: [main]

jobs:
 build:
 runs-on: ubuntu-latest
 steps:
 - uses: actions/checkout@v2
 - uses: azure/login@v1
 with:
 creds: ${{ secrets.AZURE_CREDENTIALS }}
```

5. Use the Azure CLI action to upload your code to blob storage and to purge your CDN endpoint. For `az storage blob upload-batch`, replace the placeholder with your storage account name. The script will upload to the `$web` container. For `az cdn endpoint purge`, replace the placeholders with your CDN profile name, CDN endpoint name, and resource group. To speed up your CDN purge, you can add the `--no-wait` option to `az cdn endpoint purge`. To enhance security, you can also add the `--account-key` option with your [storage account key](#).

```
- name: Upload to blob storage
 uses: azure/CLI@v1
 with:
 inlineScript: |
 az storage blob upload-batch --account-name <STORAGE_ACCOUNT_NAME> --auth-mode key -d
 '$web' -s .
 - name: Purge CDN endpoint
 uses: azure/CLI@v1
 with:
 inlineScript: |
 az cdn endpoint purge --content-paths "/" --profile-name "CDN_PROFILE_NAME" --name
 "CDN_ENDPOINT" --resource-group "RESOURCE_GROUP"
```

6. Complete your workflow by adding an action to logout of Azure. Here is the completed workflow. The file will appear in the `.github/workflows` folder of your repository.

```

name: Blob storage website CI

on:
 push:
 branches: [main]

jobs:
 build:
 runs-on: ubuntu-latest
 steps:
 - uses: actions/checkout@v2
 - uses: azure/login@v1
 with:
 creds: ${{ secrets.AZURE_CREDENTIALS }}

 - name: Upload to blob storage
 uses: azure/CLI@v1
 with:
 inlineScript: |
 az storage blob upload-batch --account-name <STORAGE_ACCOUNT_NAME> --auth-mode key -d
'$web' -s .
 - name: Purge CDN endpoint
 uses: azure/CLI@v1
 with:
 inlineScript: |
 az cdn endpoint purge --content-paths "/" --profile-name "CDN_PROFILE_NAME" --name
"CDN_ENDPOINT" --resource-group "RESOURCE_GROUP"

Azure logout
 - name: logout
 run: |
 az logout
 if: always()

```

## Review your deployment

1. Go to **Actions** for your GitHub repository.
2. Open the first result to see detailed logs of your workflow's run.

The screenshot shows the GitHub Actions interface. At the top, there are buttons for 'Workflows' (selected), 'New workflow', and 'All workflows' (highlighted in blue). Below this, there is a search bar with the placeholder 'Filter workflows'. A section titled '5 results' shows a single item: 'Update main.yml updating CDN purge' (status: green checkmark), which was triggered by 'Static Site CI #5: Commit fee09ae pushed by juliakm'.

## Clean up resources

When your static website and GitHub repository are no longer needed, clean up the resources you deployed by deleting the resource group and your GitHub repository.

## Next steps

[Learn about Azure Static Web Apps](#)

# Map a custom domain to an Azure Blob Storage endpoint

8/22/2022 • 12 minutes to read • [Edit Online](#)

You can map a custom domain to a blob service endpoint or a [static website](#) endpoint.

## NOTE

This mapping works only for subdomains (for example: `www.contoso.com`). If you want your web endpoint to be available on the root domain (for example: `contoso.com`), then you'll have to use Azure CDN. For guidance, see the [Map a custom domain with HTTPS enabled](#) section of this article. Because you're going to that section of this article to enable the root domain of your custom domain, the step within that section for enabling HTTPS is optional.

## Map a custom domain with only HTTP enabled

This approach is easier, but enables only HTTP access. If the storage account is configured to [require secure transfer](#) over HTTPS, then you must enable HTTPS access for your custom domain.

To enable HTTPS access, see the [Map a custom domain with HTTPS enabled](#) section of this article.

### Map a custom domain

#### IMPORTANT

Your custom domain will be briefly unavailable to users while you complete the configuration. If your domain currently supports an application with a service-level agreement (SLA) that requires zero downtime, then follow the steps in the [Map a custom domain with zero downtime](#) section of this article to ensure that users can access your domain while the DNS mapping takes place.

If you are unconcerned that the domain is briefly unavailable to your users, follow these steps.

- ✓ Step 1: Get the host name of your storage endpoint.
- ✓ Step 2: Create a canonical name (CNAME) record with your domain provider.
- ✓ Step 3: Register the custom domain with Azure.
- ✓ Step 4: Test your custom domain.

#### Step 1: Get the host name of your storage endpoint

The host name is the storage endpoint URL without the protocol identifier and the trailing slash.

1. In the [Azure portal](#), go to your storage account.
2. In the menu pane, under **Settings**, select **Endpoints**.
3. Copy the value of the **Blob service** endpoint or the **Static website** endpoint to a text file.

**NOTE**

The Data Lake storage endpoint is not supported (For example:

`https://mystorageaccount.dfs.core.windows.net/` ).

4. Remove the protocol identifier (For example: `HTTPS`) and the trailing slash from that string. The following table contains examples.

TYPE OF ENDPOINT	ENDPOINT	HOST NAME
blob service	<code>https://mystorageaccount.blob.core.</code>	<code>mystorageaccount.blob.core.windows.net</code>
static website	<code>https://mystorageaccount.z5.web.cor</code>	<code>mystorageaccount.z5.web.core.windows.net</code>

Set this value aside for later.

**Step 2: Create a canonical name (CNAME) record with your domain provider**

Create a CNAME record to point to your host name. A CNAME record is a type of Domain Name System (DNS) record that maps a source domain name to a destination domain name.

1. Sign in to your domain registrar's website, and then go to the page for managing DNS setting.

You might find the page in a section named **Domain Name, DNS, or Name Server Management**.

2. Find the section for managing CNAME records.

You might have to go to an advanced settings page and look for **CNAME, Alias, or Subdomains**.

3. Create a CNAME record. As part of that record, provide the following items:

- The subdomain alias such as `www` or `photos`. The subdomain is required, root domains are not supported.
- The host name that you obtained in the [Get the host name of your storage endpoint](#) section earlier in this article.

**Step 3: Register your custom domain with Azure**

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

1. In the [Azure portal](#), go to your storage account.
2. In the menu pane, under **Security + networking**, select **Networking**.
3. In the **Networking** page, choose the **Custom domain** tab.

**NOTE**

This option does not appear in accounts that have the hierarchical namespace feature enabled. For those accounts, use either PowerShell or the Azure CLI to complete this step.

4. In the **Domain name** text box, enter the name of your custom domain, including the subdomain.

For example, if your domain is *contoso.com* and your subdomain alias is *www*, enter `www.contoso.com`. If

your subdomain is *photos*, enter `photos.contoso.com`.

5. To register the custom domain, choose the **Save** button.

After the CNAME record has propagated through the Domain Name Servers (DNS), and if your users have the appropriate permissions, they can view blob data by using the custom domain.

#### Step 4: Test your custom domain

To confirm that your custom domain is mapped to your blob service endpoint, create a blob in a public container within your storage account. Then, in a web browser, access the blob by using a URI in the following format: `http://<subdomain.customdomain>/<mycontainer>/<myblob>`

For example, to access a web form in the `myforms` container in the *photos.contoso.com* custom subdomain, you might use the following URI: `http://photos.contoso.com/myforms/applicationform.htm`

### Map a custom domain with zero downtime

#### NOTE

If you are unconcerned that the domain is briefly unavailable to your users, then consider using the steps in the [Map a custom domain](#) section of this article. It's a simpler approach with fewer steps.

If your domain currently supports an application with a service-level agreement (SLA) that requires zero downtime, then follow these steps to ensure that users can access your domain while the DNS mapping takes place.

- ✓ Step 1: Get the host name of your storage endpoint.
- ✓ Step 2: Create an intermediary canonical name (CNAME) record with your domain provider.
- ✓ Step 3: Pre-register the custom domain with Azure.
- ✓ Step 4: Create a CNAME record with your domain provider.
- ✓ Step 5: Test your custom domain.

#### Step 1: Get the host name of your storage endpoint

The host name is the storage endpoint URL without the protocol identifier and the trailing slash.

1. In the [Azure portal](#), go to your storage account.
2. In the menu pane, under **Settings**, select **Endpoints**.
3. Copy the value of the **Blob service** endpoint or the **Static website** endpoint to a text file.

#### NOTE

The Data Lake storage endpoint is not supported (For example:  
`https://mystorageaccount.dfs.core.windows.net/` ).

4. Remove the protocol identifier (For example: `HTTPS`) and the trailing slash from that string. The following table contains examples.

TYPE OF ENDPOINT	ENDPOINT	HOST NAME
blob service	<code>https://mystorageaccount.blob.core.</code>	<code>mystorageaccount.blob.core.windows.net</code>

TYPE OF ENDPOINT	ENDPOINT	HOST NAME
static website	<code>https://mystorageaccount.z5.web.core.windows.net</code>	

Set this value aside for later.

#### Step 2: Create an intermediary canonical name (CNAME) record with your domain provider

Create a temporary CNAME record to point to your host name. A CNAME record is a type of DNS record that maps a source domain name to a destination domain name.

1. Sign in to your domain registrar's website, and then go to the page for managing DNS setting.

You might find the page in a section named **Domain Name, DNS, or Name Server Management**.

2. Find the section for managing CNAME records.

You might have to go to an advanced settings page and look for **CNAME, Alias, or Subdomains**.

3. Create a CNAME record. As part of that record, provide the following items:

- The subdomain alias such as `www` or `photos`. The subdomain is required, root domains are not supported.

Add the `asverify` subdomain to the alias. For example: `asverify.www` or `asverify.photos`.

- The host name that you obtained in the [Get the host name of your storage endpoint](#) section earlier in this article.

Add the subdomain `asverify` to the host name. For example:

`asverify.mystorageaccount.blob.core.windows.net`.

#### Step 3: Pre-register your custom domain with Azure

When you pre-register your custom domain with Azure, you permit Azure to recognize your custom domain without having to modify the DNS record for the domain. That way, when you do modify the DNS record for the domain, it will be mapped to the blob endpoint with no downtime.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

1. In the [Azure portal](#), go to your storage account.
2. In the menu pane, under **Security + networking**, select **Networking**.
3. In the **Networking** page, choose the **Custom domain** tab.

#### NOTE

This option does not appear in accounts that have the hierarchical namespace feature enabled. For those accounts, use either PowerShell or the Azure CLI to complete this step.

4. In the **Domain name** text box, enter the name of your custom domain, including the subdomain.

For example, if your domain is `contoso.com` and your subdomain alias is `www`, enter `www.contoso.com`. If your subdomain is `photos`, enter `photos.contoso.com`.

5. Select the **Use indirect CNAME validation** check box.
6. To register the custom domain, choose the **Save** button.

If the registration is successful, the portal notifies you that your storage account was successfully updated. Your custom domain has been verified by Azure, but traffic to your domain is not yet being routed to your storage account until you create a CNAME record with your domain provider. You'll do that in the next section.

#### **Step 4: Create a CNAME record with your domain provider**

Create a temporary CNAME record to point to your host name.

1. Sign in to your domain registrar's website, and then go to the page for managing DNS setting.

You might find the page in a section named **Domain Name, DNS, or Name Server Management**.

2. Find the section for managing CNAME records.

You might have to go to an advanced settings page and look for **CNAME, Alias, or Subdomains**.

3. Create a CNAME record. As part of that record, provide the following items:

- The subdomain alias such as `www` or `photos`. The subdomain is required, root domains are not supported.
- The host name that you obtained in the [Get the host name of your storage endpoint](#) section earlier in this article.

#### **Step 5: Test your custom domain**

To confirm that your custom domain is mapped to your blob service endpoint, create a blob in a public container within your storage account. Then, in a web browser, access the blob by using a URI in the following format: `http://<subdomain.customdomain>/<mycontainer>/<myblob>`

For example, to access a web form in the `myforms` container in the `photos.contoso.com` custom subdomain, you might use the following URI: `http://photos.contoso.com/myforms/applicationform.htm`

#### **Remove a custom domain mapping**

To remove a custom domain mapping, deregister the custom domain. Use one of the following procedures.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

1. In the [Azure portal](#), go to your storage account.
2. In the menu pane, under **Security + networking**, select **Networking**.
3. In the **Networking** page, choose the **Custom domain** tab.
4. Clear the contents of the text box that contains your custom domain name.
5. Select the **Save** button.

After the custom domain has been removed successfully, you will see a portal notification that your storage account was successfully updated.

## **Map a custom domain with HTTPS enabled**

This approach involves more steps, but it enables HTTPS access.

If you don't need users to access your blob or web content by using HTTPS, then see the [Map a custom domain with only HTTP enabled](#) section of this article.

1. Enable [Azure CDN](#) on your blob or web endpoint.

For a Blob Storage endpoint, see [Integrate an Azure storage account with Azure CDN](#).

For a static website endpoint, see [Integrate a static website with Azure CDN](#).

2. [Map Azure CDN content to a custom domain](#).

3. [Enable HTTPS on an Azure CDN custom domain](#).

**NOTE**

When you update your static website, be sure to clear cached content on the CDN edge servers by purging the CDN endpoint. For more information, see [Purge an Azure CDN endpoint](#).

4. (Optional) Review the following guidance:

- [Shared access signature \(SAS\) tokens with Azure CDN](#).
- [HTTP-to-HTTPS redirection with Azure CDN](#).
- [Pricing and billing when using Blob Storage with Azure CDN](#).

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## Next steps

- [Learn about static website hosting in Azure Blob storage](#)

# Quickstart: Route storage events to web endpoint with Azure CLI

8/22/2022 • 5 minutes to read • [Edit Online](#)

Azure Event Grid is an eventing service for the cloud. In this article, you use the Azure CLI to subscribe to Blob storage events, and trigger the event to view the result.

Typically, you send events to an endpoint that processes the event data and takes actions. However, to simplify this article, you send the events to a web app that collects and displays the messages.

When you complete the steps described in this article, you see that the event data has been sent to the web app.

# Azure Event Grid Viewer

If you don't have an [Azure subscription](#), create an [Azure free account](#) before you begin.

## Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see [Azure Cloud Shell Quickstart - Bash](#).

**A** Launch Cloud Shell

- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see [How to run the Azure CLI in a Docker container](#).
    - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options,

see [Sign in with the Azure CLI](#).

- When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
- Run `az version` to find the version and dependent libraries that are installed. To upgrade to the latest version, run `az upgrade`.
- This article requires version 2.0.70 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

## Create a resource group

Event Grid topics are Azure resources, and must be placed in an Azure resource group. The resource group is a logical collection into which Azure resources are deployed and managed.

Create a resource group with the `az group create` command.

The following example creates a resource group named `<resource_group_name>` in the *westcentralus* location. Replace `<resource_group_name>` with a unique name for your resource group.

```
az group create --name <resource_group_name> --location westcentralus
```

## Create a storage account

Blob storage events are available in general-purpose v2 storage accounts and Blob storage accounts. **General-purpose v2** storage accounts support all features for all storage services, including Blobs, Files, Queues, and Tables. A **Blob storage account** is a specialized storage account for storing your unstructured data as blobs (objects) in Azure Storage. Blob storage accounts are like general-purpose storage accounts and share all the great durability, availability, scalability, and performance features that you use today including 100% API consistency for block blobs and append blobs. For more information, see [Azure storage account overview](#).

Replace `<storage_account_name>` with a unique name for your storage account, and `<resource_group_name>` with the resource group you created earlier.

```
az storage account create \
 --name <storage_account_name> \
 --location westcentralus \
 --resource-group <resource_group_name> \
 --sku Standard_LRS \
 --kind BlobStorage \
 --access-tier Hot
```

## Create a message endpoint

Before subscribing to the topic, let's create the endpoint for the event message. Typically, the endpoint takes actions based on the event data. To simplify this quickstart, you deploy a [pre-built web app](#) that displays the event messages. The deployed solution includes an App Service plan, an App Service web app, and source code from GitHub.

Replace `<your-site-name>` with a unique name for your web app. The web app name must be unique because it's part of the DNS entry.

```
sitename=<your-site-name>

az deployment group create \
--resource-group <resource_group_name> \
--template-uri "https://raw.githubusercontent.com/Azure-Samples/azure-event-grid-
viewer/master/azuredeploy.json" \
--parameters siteName=$sitename hostingPlanName=viewerhost
```

The deployment may take a few minutes to complete. After the deployment has succeeded, view your web app to make sure it's running. In a web browser, navigate to: <https://<your-site-name>.azurewebsites.net>

You should see the site with no messages currently displayed.

## Enable the Event Grid resource provider

If you haven't previously used Event Grid in your Azure subscription, you might need to register the Event Grid resource provider. Run the following command to register the provider:

```
az provider register --namespace Microsoft.EventGrid
```

It might take a moment for the registration to finish. To check the status, run:

```
az provider show --namespace Microsoft.EventGrid --query "registrationState"
```

When `registrationState` is `Registered`, you're ready to continue.

## Subscribe to your storage account

You subscribe to a topic to tell Event Grid which events you want to track and where to send those events. The following example subscribes to the storage account you created, and passes the URL from your web app as the endpoint for event notification. Replace `<event_subscription_name>` with a name for your event subscription. For `<resource_group_name>` and `<storage_account_name>`, use the values you created earlier.

The endpoint for your web app must include the suffix `/api/updates/`.

```
storageid=$(az storage account show --name <storage_account_name> --resource-group <resource_group_name> --
query id --output tsv)
endpoint=https://$sitename.azurewebsites.net/api/updates

az eventgrid event-subscription create \
--source-resource-id $storageid \
--name <event_subscription_name> \
--endpoint $endpoint
```

View your web app again, and notice that a subscription validation event has been sent to it. Select the eye icon to expand the event data. Event Grid sends the validation event so the endpoint can verify that it wants to receive event data. The web app includes code to validate the subscription.



## Azure Event Grid Viewer

### Event Type



Microsoft.EventGrid.SubscriptionValidationEvent

```
[{
 "id": "803f7b19-15d9-4753-aad4-feaf6b535adc",
 "topic": "/subscriptions/{subscription-id}/resourceGroups/gridresourcegroup/providers/microsoft.eventgrid/",
 "subject": "",
 "data": {
 "validationCode": "2F6D57C8-97A3-4073-A4AF-3EF5DE46A8B4"
 },
 "eventType": "Microsoft.EventGrid.SubscriptionValidationEvent",
 "eventTime": "2018-05-24T17:17:44.3039911Z",
 "metadataVersion": "1",
 "dataVersion": "2"
}]
```

## Trigger an event from Blob storage

Now, let's trigger an event to see how Event Grid distributes the message to your endpoint. First, let's configure the name and key for the storage account, then we create a container, then create and upload a file. Again, use the values for `<storage_account_name>` and `<resource_group_name>` you created earlier.

```
export AZURE_STORAGE_ACCOUNT=<storage_account_name>
export AZURE_STORAGE_KEY="$(az storage account keys list --account-name <storage_account_name> --resource-group <resource_group_name> --query "[0].value" --output tsv)"

az storage container create --name testcontainer

touch testfile.txt
az storage blob upload --file testfile.txt --container-name testcontainer --name testfile.txt
```

You've triggered the event, and Event Grid sent the message to the endpoint you configured when subscribing. View your web app to see the event you just sent.

```
[{
 "topic": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxx/resourceGroups/myrg/providers/Microsoft.Storage/storageAccounts/myblobstorageaccount",
 "subject": "/blobServices/default/containers/testcontainer/blobs/testfile.txt",
 "eventType": "Microsoft.Storage.BlobCreated",
 "eventTime": "2017-08-16T20:33:51.059575Z",
 "id": "4d96b1d4-0001-00b3-58ce-16568c064fab",
 "data": {
 "api": "PutBlockList",
 "clientRequestId": "d65ca2e2-a168-4155-b7a4-2c925c18902f",
 "requestId": "4d96b1d4-0001-00b3-58ce-16568c000000",
 "eTag": "0x8D4E4E61AE038AD",
 "contentType": "text/plain",
 "contentLength": 0,
 "blobType": "BlockBlob",
 "url": "https://myblobstorageaccount.blob.core.windows.net/testcontainer/testfile.txt",
 "sequencer": "000000000000EB0000000000046199",
 "storageDiagnostics": {
 "batchId": "dffea416-b46e-4613-ac19-0371c0c5e352"
 }
 },
 "dataVersion": "",
 "metadataVersion": "1"
}]
```

## Clean up resources

If you plan to continue working with this storage account and event subscription, do not clean up the resources created in this article. If you do not plan to continue, use the following command to delete the resources you created in this article.

Replace <resource\_group\_name> with the resource group you created above.

```
az group delete --name <resource_group_name>
```

## Next steps

Now that you know how to create topics and event subscriptions, learn more about Blob storage Events and what Event Grid can help you do:

- [Reacting to Blob storage events](#)
- [About Event Grid](#)
- [Run an Azure Function in response to a blob rehydration event](#)

# Get started with AzCopy

8/22/2022 • 6 minutes to read • [Edit Online](#)

AzCopy is a command-line utility that you can use to copy blobs or files to or from a storage account. This article helps you download AzCopy, connect to your storage account, and then transfer data.

## NOTE

AzCopy **V10** is the currently supported version of AzCopy.

If you need to use a previous version of AzCopy, see the [Use the previous version of AzCopy](#) section of this article.

## Download AzCopy

First, download the AzCopy V10 executable file to any directory on your computer. AzCopy V10 is just an executable file, so there's nothing to install.

- [Windows 64-bit \(zip\)](#)
- [Windows 32-bit \(zip\)](#)
- [Linux x86-64 \(tar\)](#)
- [Linux ARM64 Preview \(tar\)](#)
- [macOS \(zip\)](#)

These files are compressed as a zip file (Windows and Mac) or a tar file (Linux). To download and decompress the tar file on Linux, see the documentation for your Linux distribution.

For detailed information on AzCopy releases see the [AzCopy release page](#).

## NOTE

If you want to copy data to and from your [Azure Table storage](#) service, then install [AzCopy version 7.3](#).

## Run AzCopy

For convenience, consider adding the directory location of the AzCopy executable to your system path for ease of use. That way you can type `azcopy` from any directory on your system.

If you choose not to add the AzCopy directory to your path, you'll have to change directories to the location of your AzCopy executable and type `azcopy` or `.\azcopy` in Windows PowerShell command prompts.

As an owner of your Azure Storage account, you aren't automatically assigned permissions to access data. Before you can do anything meaningful with AzCopy, you need to decide how you'll provide authorization credentials to the storage service.

## Authorize AzCopy

You can provide authorization credentials by using Azure Active Directory (AD), or by using a Shared Access Signature (SAS) token.

Use this table as a guide:

STORAGE TYPE	CURRENTLY SUPPORTED METHOD OF AUTHORIZATION
Blob storage	Azure AD & SAS
Blob storage (hierarchical namespace)	Azure AD & SAS
File storage	SAS only

#### Option 1: Use Azure Active Directory

This option is available for blob Storage only. By using Azure Active Directory, you can provide credentials once instead of having to append a SAS token to each command.

#### Option 2: Use a SAS token

You can append a SAS token to each source or destination URL that use in your AzCopy commands.

This example command recursively copies data from a local directory to a blob container. A fictitious SAS token is appended to the end of the container URL.

```
azcopy copy "C:\local\path" "https://account.blob.core.windows.net/mycontainer1/?sv=2018-03-28&ss=bjqt&srt=sco&sp=rwddgcup&se=2019-05-01T05:01:17Z&st=2019-04-30T21:01:17Z&spr=https&sig=MGCXiyezbttkr3ewJih2AR8Krghsy1DGM9ovN734bQF4%3D" --recursive=true
```

To learn more about SAS tokens and how to obtain one, see [Using shared access signatures \(SAS\)](#).

#### NOTE

The [Secure transfer required](#) setting of a storage account determines whether the connection to a storage account is secured with Transport Layer Security (TLS). This setting is enabled by default.

## Transfer data

After you've authorized your identity or obtained a SAS token, you can begin transferring data.

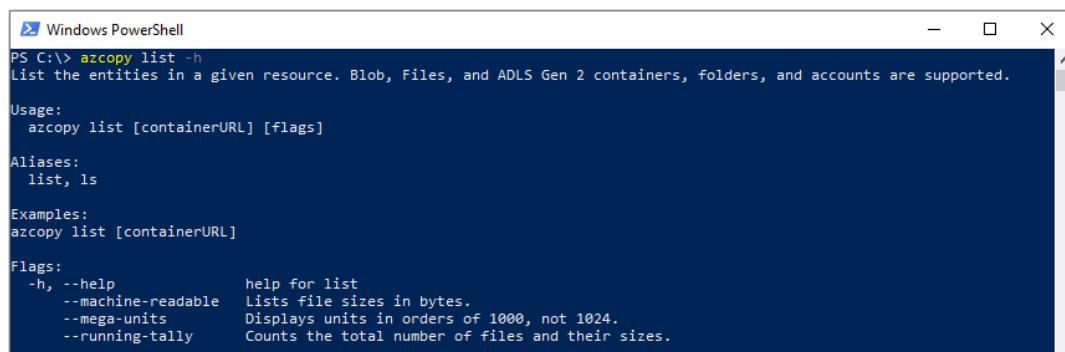
To find example commands, see any of these articles.

SERVICE	ARTICLE
Azure Blob Storage	<a href="#">Upload files to Azure Blob Storage</a>
Azure Blob Storage	<a href="#">Download blobs from Azure Blob Storage</a>
Azure Blob Storage	<a href="#">Copy blobs between Azure storage accounts</a>
Azure Blob Storage	<a href="#">Synchronize with Azure Blob Storage</a>
Azure Files	<a href="#">Transfer data with AzCopy and file storage</a>
Amazon S3	<a href="#">Copy data from Amazon S3 to Azure Storage</a>
Google Cloud Storage	<a href="#">Copy data from Google Cloud Storage to Azure Storage (preview)</a>
Azure Stack storage	<a href="#">Transfer data with AzCopy and Azure Stack storage</a>

## Get command help

To see a list of commands, type `azcopy -h` and then press the ENTER key.

To learn about a specific command, just include the name of the command (For example: `azcopy list -h`).



```
PS C:\> azcopy list -h
List the entities in a given resource. Blob, Files, and ADLS Gen 2 containers, folders, and accounts are supported.

Usage:
 azcopy list [containerURL] [flags]

Aliases:
 list, ls

Examples:
azcopy list [containerURL]

Flags:
 -h, --help help for list
 --machine-readable Lists file sizes in bytes.
 --mega-units Displays units in orders of 1000, not 1024.
 --running-tally Counts the total number of files and their sizes.
```

## List of commands

The following table lists all AzCopy v10 commands. Each command links to a reference article.

COMMAND	DESCRIPTION
<a href="#">azcopy bench</a>	Runs a performance benchmark by uploading or downloading test data to or from a specified location.
<a href="#">azcopy copy</a>	Copies source data to a destination location
<a href="#">azcopy doc</a>	Generates documentation for the tool in Markdown format.
<a href="#">azcopy env</a>	Shows the environment variables that can configure AzCopy's behavior.
<a href="#">azcopy jobs</a>	Subcommands related to managing jobs.
<a href="#">azcopy jobs clean</a>	Remove all log and plan files for all jobs.
<a href="#">azcopy jobs list</a>	Displays information on all jobs.
<a href="#">azcopy jobs remove</a>	Remove all files associated with the given job ID.
<a href="#">azcopy jobs resume</a>	Resumes the existing job with the given job ID.
<a href="#">azcopy jobs show</a>	Shows detailed information for the given job ID.
<a href="#">azcopy list</a>	Lists the entities in a given resource.
<a href="#">azcopy login</a>	Logs in to Azure Active Directory to access Azure Storage resources.
<a href="#">azcopy login status</a>	Lists the entities in a given resource.
<a href="#">azcopy logout</a>	Logs the user out and terminates access to Azure Storage resources.
<a href="#">azcopy make</a>	Creates a container or file share.
<a href="#">azcopy remove</a>	Delete blobs or files from an Azure storage account.
<a href="#">azcopy sync</a>	Replicates the source location to the destination location.

### NOTE

AzCopy does not have a command to rename files.

## Use in a script

### Obtain a static download link

Over time, the AzCopy [download link](#) will point to new versions of AzCopy. If your script downloads AzCopy, the script might stop working if a newer version of AzCopy modifies features that your script depends upon.

To avoid these issues, obtain a static (unchanging) link to the current version of AzCopy. That way, your script downloads the same exact version of AzCopy each time that it runs.

To obtain the link, run this command:

OPERATING SYSTEM	COMMAND
Linux	<pre>curl -s -D- https://aka.ms/downloadazcopy-v10-linux   grep ^Location</pre>

OPERATING SYSTEM	COMMAND
Windows (PowerShell Core 7)	(Invoke-WebRequest https://aka.ms/downloadazcopy-v10-windows -MaximumRedirection 0 -ErrorAction silentlycontinue -SkipHttpErrorCheck).headers.location[0]
Windows (PowerShell 5.1)	(Invoke-WebRequest https://aka.ms/downloadazcopy-v10-windows -MaximumRedirection 0 -ErrorAction silentlycontinue ).headers.location

#### NOTE

For Linux, `--strip-components=1` on the `tar` command removes the top-level folder that contains the version name, and instead extracts the binary directly into the current folder. This allows the script to be updated with a new version of `azcopy` by only updating the `wget` URL.

The URL appears in the output of this command. Your script can then download AzCopy by using that URL.

OPERATING SYSTEM	COMMAND
Linux	wget -O azcopy_v10.tar.gz https://aka.ms/downloadazcopy-v10-linux && tar -xvf azcopy_v10.tar.gz --strip-components=1
Windows	Invoke-WebRequest https://azcopyvnext.azureedge.net/release20190517/azcopy_windows_amd64_1 -Outfile azcopyv10.zip <<Unzip here>>

#### Escape special characters in SAS tokens

In batch files that have the `.cmd` extension, you'll have to escape the `%` characters that appear in SAS tokens. You can do that by adding an additional `%` character next to existing `%` characters in the SAS token string.

#### Run scripts by using Jenkins

If you plan to use [Jenkins](#) to run scripts, make sure to place the following command at the beginning of the script.

```
/usr/bin/keyctl new_session
```

## Use in Azure Storage Explorer

[Storage Explorer](#) uses AzCopy to perform all of its data transfer operations. You can use [Storage Explorer](#) if you want to leverage the performance advantages of AzCopy, but you prefer to use a graphical user interface rather than the command line to interact with your files.

Storage Explorer uses your account key to perform operations, so after you sign into Storage Explorer, you won't need to provide additional authorization credentials.

## Configure, optimize, and fix

See any of the following resources:

- [AzCopy configuration settings](#)
- [Optimize the performance of AzCopy](#)
- [Find errors and resume jobs by using log and plan files in AzCopy](#)
- [Troubleshoot problems with AzCopy v10](#)

## Use a previous version

If you need to use the previous version of AzCopy, see either of the following links:

- [AzCopy on Windows \(v8\)](#)

- [AzCopy on Linux \(v7\)](#)

## Next steps

If you have questions, issues, or general feedback, submit them [on GitHub](#) page.

# Authorize access to blobs with AzCopy and Azure Active Directory (Azure AD)

8/22/2022 • 10 minutes to read • [Edit Online](#)

You can provide AzCopy with authorization credentials by using Azure AD. That way, you won't have to append a shared access signature (SAS) token to each command.

Start by verifying your role assignments. Then, choose what type of *security principal* you want to authorize. A [user identity](#), a [managed identity](#), and a [service principal](#) are each a type of security principal.

A user identity is any user that has an identity in Azure AD. It's the easiest security principal to authorize. Managed identities and service principals are great options if you plan to use AzCopy inside of a script that runs without user interaction. A managed identity is better suited for scripts that run from an Azure Virtual Machine (VM), and a service principal is better suited for scripts that run on-premises.

For more information about AzCopy, [Get started with AzCopy](#).

## Verify role assignments

The level of authorization that you need is based on whether you plan to upload files or just download them.

If you just want to download files, then verify that the [Storage Blob Data Reader](#) role has been assigned to your user identity, managed identity, or service principal.

If you want to upload files, then verify that one of these roles has been assigned to your security principal:

- [Storage Blob Data Contributor](#)
- [Storage Blob Data Owner](#)

These roles can be assigned to your security principal in any of these scopes:

- Container (file system)
- Storage account
- Resource group
- Subscription

To learn how to verify and assign roles, see [Assign an Azure role for access to blob data](#).

### NOTE

Keep in mind that Azure role assignments can take up to five minutes to propagate.

You don't need to have one of these roles assigned to your security principal if your security principal is added to the access control list (ACL) of the target container or directory. In the ACL, your security principal needs write permission on the target directory, and execute permission on container and each parent directory.

To learn more, see [Access control model in Azure Data Lake Storage Gen2](#).

## Authorize a user identity

After you've verified that your user identity has been given the necessary authorization level, open a command prompt, type the following command, and then press the ENTER key.

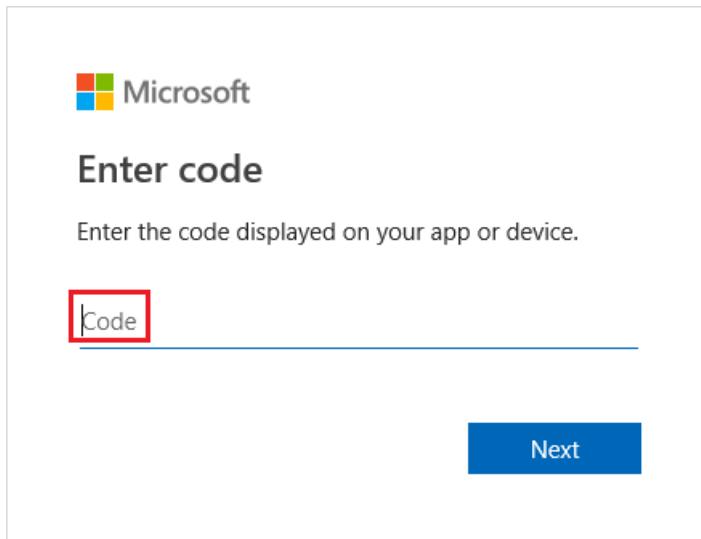
```
azcopy login
```

If you receive an error, try including the tenant ID of the organization to which the storage account belongs.

```
azcopy login --tenant-id=<tenant-id>
```

Replace the `<tenant-id>` placeholder with the tenant ID of the organization to which the storage account belongs. To find the tenant ID, select **Azure Active Directory > Properties > Directory ID** in the Azure portal.

This command returns an authentication code and the URL of a website. Open the website, provide the code, and then choose the **Next** button.



A sign-in window will appear. In that window, sign into your Azure account by using your Azure account credentials. After you've successfully signed in, you can close the browser window and begin using AzCopy.

## Authorize a managed identity

This is a great option if you plan to use AzCopy inside of a script that runs without user interaction, and the script runs from an Azure Virtual Machine (VM). When using this option, you won't have to store any credentials on the VM.

You can sign into your account by using a system-wide managed identity that you've enabled on your VM, or by using the client ID, Object ID, or Resource ID of a user-assigned managed identity that you've assigned to your VM.

To learn more about how to enable a system-wide managed identity or create a user-assigned managed identity, see [Configure managed identities for Azure resources on a VM using the Azure portal](#).

### Authorize by using a system-wide managed identity

First, make sure that you've enabled a system-wide managed identity on your VM. See [System-assigned managed identity](#).

Then, in your command console, type the following command, and then press the ENTER key.

```
azcopy login --identity
```

### Authorize by using a user-assigned managed identity

First, make sure that you've enabled a user-assigned managed identity on your VM. See [User-assigned managed](#)

identity.

Then, in your command console, type any of the following commands, and then press the ENTER key.

```
azcopy login --identity --identity-client-id "<client-id>"
```

Replace the `<client-id>` placeholder with the client ID of the user-assigned managed identity.

```
azcopy login --identity --identity-object-id "<object-id>"
```

Replace the `<object-id>` placeholder with the object ID of the user-assigned managed identity.

```
azcopy login --identity --identity-resource-id "<resource-id>"
```

Replace the `<resource-id>` placeholder with the resource ID of the user-assigned managed identity.

## Authorize a service principal

This is a great option if you plan to use AzCopy inside of a script that runs without user interaction, particularly when running on-premises. If you plan to run AzCopy on VMs that run in Azure, a managed service identity is easier to administer. To learn more, see the [Authorize a managed identity](#) section of this article.

Before you run a script, you have to sign in interactively at least one time so that you can provide AzCopy with the credentials of your service principal. Those credentials are stored in a secured and encrypted file so that your script doesn't have to provide that sensitive information.

You can sign into your account by using a client secret or by using the password of a certificate that is associated with your service principal's app registration.

To learn more about creating service principal, see [How to: Use the portal to create an Azure AD application and service principal that can access resources](#).

To learn more about service principals in general, see [Application and service principal objects in Azure Active Directory](#)

### Authorize a service principal by using a client secret

Start by setting the `AZCOPY_SPA_CLIENT_SECRET` environment variable to the client secret of your service principal's app registration.

#### NOTE

Make sure to set this value from your command prompt, and not in the environment variable settings of your operating system. That way, the value is available only to the current session.

This example shows how you could do this in PowerShell.

```
$env:AZCOPY_SPA_CLIENT_SECRET="$(Read-Host -prompt "Enter key")"
```

#### NOTE

Consider using a prompt as shown in this example. That way, your password won't appear in your console's command history.

Next, type the following command, and then press the ENTER key.

```
azcopy login --service-principal --application-id application-id --tenant-id=tenant-id
```

Replace the `<application-id>` placeholder with the application ID of your service principal's app registration. Replace the `<tenant-id>` placeholder with the tenant ID of the organization to which the storage account belongs. To find the tenant ID, select **Azure Active Directory > Properties > Directory ID** in the Azure portal.

#### Authorize a service principal by using a certificate

If you prefer to use your own credentials for authorization, you can upload a certificate to your app registration, and then use that certificate to log in.

In addition to uploading your certificate to your app registration, you'll also need to have a copy of the certificate saved to the machine or VM where AzCopy will be running. This copy of the certificate should be in .PFX or .PEM format, and must include the private key. The private key should be password-protected. If you're using Windows, and your certificate exists only in a certificate store, make sure to export that certificate to a PFX file (including the private key). For guidance, see [Export-PfxCertificate](#)

Next, set the `AZCOPY_SPA_CERT_PASSWORD` environment variable to the certificate password.

#### NOTE

Make sure to set this value from your command prompt, and not in the environment variable settings of your operating system. That way, the value is available only to the current session.

This example shows how you could do this task in PowerShell.

```
$env:AZCOPY_SPA_CERT_PASSWORD="$(Read-Host -prompt "Enter key")"
```

Next, type the following command, and then press the ENTER key.

```
azcopy login --service-principal --certificate-path <path-to-certificate-file> --tenant-id=<tenant-id>
```

Replace the `<path-to-certificate-file>` placeholder with the relative or fully qualified path to the certificate file. AzCopy saves the path to this certificate but it doesn't save a copy of the certificate, so make sure to keep that certificate in place. Replace the `<tenant-id>` placeholder with the tenant ID of the organization to which the storage account belongs. To find the tenant ID, select **Azure Active Directory > Properties > Directory ID** in the Azure portal.

#### NOTE

Consider using a prompt as shown in this example. That way, your password won't appear in your console's command history.

## Authorize without a secret store

The `azcopy login` command retrieves an OAuth token and then places that token into a secret store on your system. If your operating system doesn't have a secret store such as a Linux *keyring*, the `azcopy login` command won't work because there is nowhere to place the token.

Instead of using the `azcopy login` command, you can set in-memory environment variables. Then run any

AzCopy command. AzCopy will retrieve the Auth token required to complete the operation. After the operation completes, the token disappears from memory.

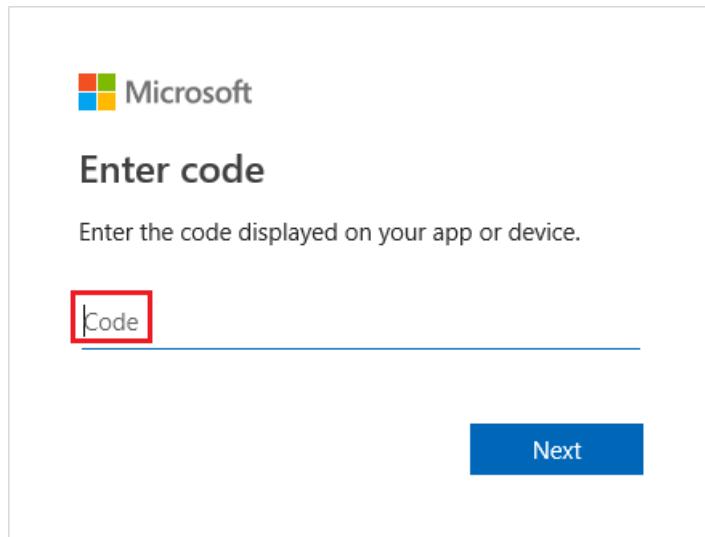
### Authorize a user identity

After you've verified that your user identity has been given the necessary authorization level, type the following command, and then press the ENTER key.

```
export AZCOPY_AUTO_LOGIN_TYPE=DEVICE
```

Then, run any azcopy command (For example: `azcopy list https://contoso.blob.core.windows.net`).

This command returns an authentication code and the URL of a website. Open the website, provide the code, and then choose the **Next** button.



A sign-in window will appear. In that window, sign into your Azure account by using your Azure account credentials. After you've successfully signed in, the operation can complete.

### Authorize by using a system-wide managed identity

First, make sure that you've enabled a system-wide managed identity on your VM. See [System-assigned managed identity](#).

Type the following command, and then press the ENTER key.

```
export AZCOPY_AUTO_LOGIN_TYPE=MSI
```

Then, run any azcopy command (For example: `azcopy list https://contoso.blob.core.windows.net`).

### Authorize by using a user-assigned managed identity

First, make sure that you've enabled a user-assigned managed identity on your VM. See [User-assigned managed identity](#).

Type the following command, and then press the ENTER key.

```
export AZCOPY_AUTO_LOGIN_TYPE=MSI
```

Then, type any of the following commands, and then press the ENTER key.

```
export AZCOPY_MSI_CLIENT_ID=<client-id>
```

Replace the `<client-id>` placeholder with the client ID of the user-assigned managed identity.

```
export AZCOPY_MSI_OBJECT_ID=<object-id>
```

Replace the `<object-id>` placeholder with the object ID of the user-assigned managed identity.

```
export AZCOPY_MSI_RESOURCE_STRING=<resource-id>
```

Replace the `<resource-id>` placeholder with the resource ID of the user-assigned managed identity.

After you set these variables, you can run any azcopy command (For example:

```
azcopy list https://contoso.blob.core.windows.net).
```

## Authorize a service principal

You can sign into your account by using a client secret or by using the password of a certificate that is associated with your service principal's app registration.

### Authorize a service principal by using a client secret

Type the following command, and then press the ENTER key.

```
export AZCOPY_AUTO_LOGIN_TYPE=SPN
export AZCOPY_SPA_APPLICATION_ID=<application-id>
export AZCOPY_SPA_CLIENT_SECRET=<client-secret>
export AZCOPY_TENANT_ID=<tenant-id>
```

Replace the `<application-id>` placeholder with the application ID of your service principal's app registration.

Replace the `<client-secret>` placeholder with the client secret. Replace the `<tenant-id>` placeholder with the tenant ID of the organization to which the storage account belongs. To find the tenant ID, select **Azure Active Directory > Properties > Directory ID** in the Azure portal.

### NOTE

Consider using a prompt to collect the password from the user. That way, your password won't appear in your command history.

Then, run any azcopy command (For example: `azcopy list https://contoso.blob.core.windows.net`).

### Authorize a service principal by using a certificate

If you prefer to use your own credentials for authorization, you can upload a certificate to your app registration, and then use that certificate to log in.

In addition to uploading your certificate to your app registration, you'll also need to have a copy of the certificate saved to the machine or VM where AzCopy will be running. This copy of the certificate should be in .PFX or .PEM format, and must include the private key. The private key should be password-protected.

Type the following command, and then press the ENTER key.

```
export AZCOPY_AUTO_LOGIN_TYPE=SPN
export AZCOPY_SPA_CERT_PATH=<path-to-certificate-file>
export AZCOPY_SPA_CERT_PASSWORD=<certificate-password>
export AZCOPY_TENANT_ID=<tenant-id>
```

Replace the `<path-to-certificate-file>` placeholder with the relative or fully qualified path to the certificate file. AzCopy saves the path to this certificate but it doesn't save a copy of the certificate, so make sure to keep that

certificate in place. Replace the `<certificate-password>` placeholder with the password of the certificate. Replace the `<tenant-id>` placeholder with the tenant ID of the organization to which the storage account belongs. To find the tenant ID, select **Azure Active Directory > Properties > Directory ID** in the Azure portal.

**NOTE**

Consider using a prompt to collect the password from the user. That way, your password won't appear in your command history.

Then, run any azcopy command (For example: `azcopy list https://contoso.blob.core.windows.net`).

## Next steps

- For more information about AzCopy, [Get started with AzCopy](#)
- If you have questions, issues, or general feedback, submit them [on GitHub page](#).

# Optimize the performance of AzCopy with Azure Storage

8/22/2022 • 6 minutes to read • [Edit Online](#)

AzCopy is a command-line utility that you can use to copy blobs or files to or from a storage account. This article helps you to optimize performance.

## NOTE

If you're looking for content to help you get started with AzCopy, see [Get started with AzCopy](#)

You can benchmark performance, and then use commands and environment variables to find an optimal tradeoff between performance and resource consumption.

## Run benchmark tests

You can run a performance benchmark test on specific blob containers or file shares to view general performance statistics and to identify performance bottlenecks. You can run the test by uploading or downloading generated test data.

Use the following command to run a performance benchmark test.

### Syntax

```
azcopy benchmark 'https://<storage-account-name>.blob.core.windows.net/<container-name>'
```

### Example

```
azcopy benchmark 'https://mystorageaccount.blob.core.windows.net/mycontainer/myBlobDirectory?sv=2018-03-28&ss=bjqt&srs=sco&sp=rjk1hjup&se=2019-05-10T04:37:48Z&st=2019-05-09T20:37:48Z&spr=https&sig=%2FSOVEFfsKDqRry4bk3qz1vAQFwY5DDzp2%2B%2F3Eykf%2FJLs%3D'
```

## TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

This command runs a performance benchmark by uploading test data to a specified destination. The test data is generated in memory, uploaded to the destination, then deleted from the destination after the test is complete. You can specify how many files to generate and what size you'd like them to be by using optional command parameters.

If you prefer to run this test by downloading data, set the `mode` parameter to `download`. For detailed reference docs, see [azcopy benchmark](#).

## Optimize for large numbers of small files

Throughput can decrease when transferring small files, especially when transferring large numbers of them. To maximize performance, reduce the size of each job. For download and upload operations, increase concurrency,

decrease log activity, and turn off features that incur high performance costs.

#### Reduce the size of each job

To achieve optimal performance, ensure that each job transfers fewer than 10 million files. Jobs that transfer more than 50 million files can perform poorly because the AzCopy job tracking mechanism incurs a significant amount of overhead. To reduce overhead, consider dividing large jobs into smaller ones.

One way to reduce the size of a job is to limit the number of files affected by a job. You can use command parameters to do that. For example, a job can copy only a subset of directories by using the `include path` parameter as part of the [azcopy copy](#) command.

Use the `include-pattern` parameter to copy files that have a specific extension (for example: `*.pdf`). In a separate job, use the `exclude-pattern` parameter to copy all files that don't have `*.pdf` extension. See [Upload specific files](#) and [Download specific blobs](#) for examples.

After you've decided how to divide large jobs into smaller ones, consider running jobs on more than one Virtual Machine (VM).

#### Increase concurrency

If you're uploading or downloading files, use the `AZCOPY_CONCURRENCY_VALUE` environment variable to increase the number of concurrent requests that can occur on your machine. Set this variable as high as possible without compromising the performance of your machine. To learn more about this variable, see the [Increase the number of concurrent requests](#) section of this article.

If you're copying blobs between storage accounts, consider setting the value of the `AZCOPY_CONCURRENCY_VALUE` environment variable to a value greater than `1000`. You can set this variable high because AzCopy uses server-to-server APIs, so data is copied directly between storage servers and does not use your machine's processing power.

#### Decrease the number of logs generated

You can improve performance by reducing the number of log entries that AzCopy creates as it completes an operation. By default, AzCopy logs all activity related to an operation. To achieve optimal performance, consider setting the `log-level` parameter of your copy, sync, or remove command to `ERROR`. That way, AzCopy logs only errors. By default, the value log level is set to `INFO`.

#### Turn off length checking

If you're uploading or downloading files, consider setting the `--check-length` of your copy and sync commands to `false`. This prevents AzCopy from verifying the length of a file after a transfer. By default, AzCopy checks the length to ensure that source and destination files match after a transfer completes. AzCopy performs this check after each file transfer. This check can degrade performance when jobs transfer large numbers of small files.

#### Turn on concurrent local scanning (Linux)

File scans on some Linux systems don't execute fast enough to saturate all of the parallel network connections. In these cases, you can set the `AZCOPY_CONCURRENT_SCAN` to a higher number.

## Increase the number of concurrent requests

You can increase throughput by setting the `AZCOPY_CONCURRENCY_VALUE` environment variable. This variable specifies the number of concurrent requests that can occur.

If your computer has fewer than 5 CPUs, then the value of this variable is set to `32`. Otherwise, the default value is equal to 16 multiplied by the number of CPUs. The maximum default value of this variable is `300`, but you can manually set this value higher or lower.

OPERATING SYSTEM	COMMAND
Windows	<code>set AZCOPY_CONCURRENCY_VALUE=&lt;value&gt;</code>
Linux	<code>export AZCOPY_CONCURRENCY_VALUE=&lt;value&gt;</code>
macOS	<code>export AZCOPY_CONCURRENCY_VALUE=&lt;value&gt;</code>

Use the `azcopy env` to check the current value of this variable. If the value is blank, then you can read which value is being used by looking at the beginning of any AzCopy log file. The selected value, and the reason it was selected, are reported there.

Before you set this variable, we recommend that you run a benchmark test. The benchmark test process will report the recommended concurrency value. Alternatively, if your network conditions and payloads vary, set this variable to the word `AUTO` instead of to a particular number. That will cause AzCopy to always run the same automatic tuning process that it uses in benchmark tests.

## Limit the throughput data rate

You can use the `cap-mbps` flag in your commands to place a ceiling on the throughput data rate. For example, the following command resumes a job and caps throughput to `10` megabits (Mb) per second.

```
azcopy jobs resume <job-id> --cap-mbps 10
```

## Optimize memory use

Set the `AZCOPY_BUFFER_GB` environment variable to specify the maximum amount of your system memory you want AzCopy to use for buffering when downloading and uploading files. Express this value in gigabytes (GB).

OPERATING SYSTEM	COMMAND
Windows	<code>set AZCOPY_BUFFER_GB=&lt;value&gt;</code>
Linux	<code>export AZCOPY_BUFFER_GB=&lt;value&gt;</code>
macOS	<code>export AZCOPY_BUFFER_GB=&lt;value&gt;</code>

### NOTE

Job tracking always incurs additional overhead in memory usage. The amount varies based on the number of transfers in a job. Buffers are the largest component of memory usage. You can help control overhead by using `AZCOPY_BUFFER_GB` to approximately meet your requirements, but there is no flag available to strictly cap the overall memory usage.

## Optimize file synchronization

The `sync` command identifies all files at the destination, and then compares file names and last modified timestamps before the starting the sync operation. If you have a large number of files, then you can improve performance by eliminating this up-front processing.

To accomplish this, use the `azcopy copy` command instead, and set the `--overwrite` flag to `ifSourceNewer`. AzCopy will compare files as they are copied without performing any up-front scans and comparisons. This

provides a performance edge in cases where there are a large number of files to compare.

The [azcopy copy](#) command doesn't delete files from the destination, so if you want to delete files at the destination when they no longer exist at the source, then use the [azcopy sync](#) command with the

`--delete-destination` flag set to a value of `true` or `prompt`.

## See also

- [Get started with AzCopy](#)

# Find errors and resume jobs by using log and plan files in AzCopy

8/22/2022 • 3 minutes to read • [Edit Online](#)

AzCopy is a command-line utility that you can use to copy blobs or files to or from a storage account. This article helps you use logs to diagnose errors, and then use plan files to resume jobs. This article also shows how to configure log and plan files by changing their verbosity level, and the default location where they are stored.

## NOTE

If you're looking for content to help you get started with AzCopy, see [Get started with AzCopy](#). This article applies to AzCopy V10 as is this is the currently supported version of AzCopy. If you need to use a previous version of AzCopy, see [Use the previous version of AzCopy](#).

## Log and plan files

AzCopy creates *log* and *plan* files for every job. You can use these logs to investigate and troubleshoot any potential problems.

The logs will contain the status of failure (`UPLOADFAILED`, `COPYFAILED`, and `DOWNLOADFAILED`), the full path, and the reason of the failure.

By default, the log and plan files are located in the `%USERPROFILE%\azcopy` directory on Windows or `$HOME$\azcopy` directory on Mac and Linux, but you can change that location.

The relevant error isn't necessarily the first error that appears in the file. For errors such as network errors, timeouts and Server Busy errors, AzCopy will retry up to 20 times and usually the retry process succeeds. The first error that you see might be something harmless that was successfully retried. So instead of looking at the first error in the file, look for the errors that are near `UPLOADFAILED`, `COPYFAILED`, or `DOWNLOADFAILED`.

## IMPORTANT

When submitting a request to Microsoft Support (or troubleshooting the issue involving any third party), share the redacted version of the command you want to execute. This ensures the SAS isn't accidentally shared with anybody. You can find the redacted version at the start of the log file.

## Review the logs for errors

The following command will get all errors with `UPLOADFAILED` status from the `04dc9ca9-158f-7945-5933-564021086c79` log:

### Windows (PowerShell)

```
Select-String UPLOADFAILED .\04dc9ca9-158f-7945-5933-564021086c79.log
```

### Linux

```
grep UPLOADFAILED .\04dc9ca9-158f-7945-5933-564021086c79.log
```

## View and resume jobs

Each transfer operation will create an AzCopy job. Use the following command to view the history of jobs:

```
azcopy jobs list
```

To view the job statistics, use the following command:

```
azcopy jobs show <job-id>
```

To filter the transfers by status, use the following command:

```
azcopy jobs show <job-id> --with-status=Failed
```

**TIP**

The value of the `--with-status` flag is case-sensitive.

Use the following command to resume a failed/canceled job. This command uses its identifier along with the SAS token as it isn't persistent for security reasons:

```
azcopy jobs resume <job-id> --source-sas=<sas-token> --destination-sas=<sas-token>
```

**TIP**

Enclose path arguments such as the SAS token with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

When you resume a job, AzCopy looks at the job plan file. The plan file lists all the files that were identified for processing when the job was first created. When you resume a job, AzCopy will attempt to transfer all of the files that are listed in the plan file which weren't already transferred.

## Change the location of plan files

Use any of these commands.

OPERATING SYSTEM	COMMAND
Windows	PowerShell: <code>\$env:AZCOPY_JOB_PLAN_LOCATION=&lt;value&gt;</code> In a command prompt use: <code>set AZCOPY_JOB_PLAN_LOCATION=&lt;value&gt;</code>
Linux	<code>export AZCOPY_JOB_PLAN_LOCATION=&lt;value&gt;</code>
macOS	<code>export AZCOPY_JOB_PLAN_LOCATION=&lt;value&gt;</code>

Use the `azcopy env` to check the current value of this variable. If the value is blank, then plan files are written to the default location.

## Change the location of log files

Use any of these commands.

OPERATING SYSTEM	COMMAND
Windows	PowerShell: <code>\$env:AZCOPY_LOG_LOCATION=&lt;value&gt;</code> In a command prompt use: <code>set AZCOPY_LOG_LOCATION=&lt;value&gt;</code>
Linux	<code>export AZCOPY_LOG_LOCATION=&lt;value&gt;</code>
macOS	<code>export AZCOPY_LOG_LOCATION=&lt;value&gt;</code>

Use the `azcopy env` to check the current value of this variable. If the value is blank, then logs are written to the default location.

## Change the default log level

By default, AzCopy log level is set to `INFO`. If you would like to reduce the log verbosity to save disk space, overwrite this setting by using the `--log-level` option.

Available log levels are: `DEBUG`, `INFO`, `WARNING`, `ERROR`, and `NONE`.

## Remove plan and log files

If you want to remove all plan and log files from your local machine to save disk space, use the `azcopy jobs clean` command.

To remove the plan and log files associated with only one job, use `azcopy jobs rm <job-id>`. Replace the `<job-id>` placeholder in this example with the job ID of the job.

## See also

- [Get started with AzCopy](#)

# Upload files to Azure Blob storage by using AzCopy

8/22/2022 • 6 minutes to read • [Edit Online](#)

You can upload files and directories to Blob storage by using the AzCopy v10 command-line utility.

To see examples for other types of tasks such as downloading blobs, synchronizing with Blob storage, or copying blobs between accounts, see the links presented in the [Next Steps](#) section of this article.

## Get started

See the [Get started with AzCopy](#) article to download AzCopy and learn about the ways that you can provide authorization credentials to the storage service.

### NOTE

The examples in this article assume that you've provided authorization credentials by using Azure Active Directory (Azure AD).

If you'd rather use a SAS token to authorize access to blob data, then you can append that token to the resource URL in each AzCopy command. For example:

```
'https://<storage-account-name>.blob.core.windows.net/<container-name><SAS-token>'.
```

## Create a container

You can use the [azcopy make](#) command to create a container.

### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

## Syntax

```
azcopy make 'https://<storage-account-name>.<blob or dfs>.core.windows.net/<container-name>'
```

## Example

```
azcopy make 'https://mystorageaccount.blob.core.windows.net/mycontainer'
```

## Example (hierarchical namespace)

```
azcopy make 'https://mystorageaccount.dfs.core.windows.net/mycontainer'
```

For detailed reference docs, see [azcopy make](#).

## Upload a file

Upload a file by using the [azcopy copy](#) command.

**TIP**

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

## Syntax

```
azcopy copy '<local-file-path>' 'https://<storage-account-name>.<blob or dfs>.core.windows.net/<container-name>/<blob-name>'
```

## Example

```
azcopy copy 'C:\myDirectory\myTextFile.txt'
'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile.txt'
```

## Example (hierarchical namespace)

```
azcopy copy 'C:\myDirectory\myTextFile.txt'
'https://mystorageaccount.dfs.core.windows.net/mycontainer/myTextFile.txt'
```

You can also upload a file by using a wildcard symbol (\*) anywhere in the file path or file name. For example:

'C:\myDirectory\\*.txt' , or C:\my\*\\*.txt .

## Upload a directory

Upload a directory by using the [azcopy copy](#) command.

This example copies a directory (and all of the files in that directory) to a blob container. The result is a directory in the container by the same name.

**TIP**

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

## Syntax

```
azcopy copy '<local-directory-path>' 'https://<storage-account-name>.<blob or
dfs>.core.windows.net/<container-name>' --recursive
```

## Example

```
azcopy copy 'C:\myDirectory' 'https://mystorageaccount.blob.core.windows.net/mycontainer' --recursive
```

## Example (hierarchical namespace)

```
azcopy copy 'C:\myDirectory' 'https://mystorageaccount.dfs.core.windows.net/mycontainer' --recursive
```

To copy to a directory within the container, just specify the name of that directory in your command string.

## Example

```
azcopy copy 'C:\myDirectory' 'https://mystorageaccount.blob.core.windows.net/mycontainer/myBlobDirectory' --recursive
```

### Example (hierarchical namespace)

```
azcopy copy 'C:\myDirectory' 'https://mystorageaccount.dfs.core.windows.net/mycontainer/myBlobDirectory' --recursive
```

If you specify the name of a directory that doesn't exist in the container, AzCopy creates a new directory by that name.

## Upload directory contents

Upload the contents of a directory by using the [azcopy copy](#) command. Use the wildcard symbol (\*) to upload the contents without copying the containing directory itself.

#### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

### Syntax

```
azcopy copy '<local-directory-path>*' 'https://<storage-account-name>.<blob or
dfs>.core.windows.net/<container-name>/<directory-path>'
```

### Example

```
azcopy copy 'C:\myDirectory*' 'https://mystorageaccount.blob.core.windows.net/mycontainer/myBlobDirectory'
```

### Example (hierarchical namespace)

```
azcopy copy 'C:\myDirectory*' 'https://mystorageaccount.dfs.core.windows.net/mycontainer/myBlobDirectory'
```

Append the `--recursive` flag to upload files in all subdirectories.

## Upload specific files

You can upload specific files by using complete file names, partial names with wildcard characters (\*), or by using dates and times.

#### TIP

These examples enclose path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

### Specify multiple complete file names

Use the [azcopy copy](#) command with the `--include-path` option. Separate individual file names by using a semicolon ( ; ).

## Syntax

```
azcopy copy '<local-directory-path>' 'https://<storage-account-name>.<blob or
dfs>.core.windows.net/<container-name>' --include-path <semicolon-separated-file-list>
```

## Example

```
azcopy copy 'C:\myDirectory' 'https://mystorageaccount.blob.core.windows.net/mycontainer' --include-path
'photos;documents\myfile.txt' --recursive'
```

## Example (hierarchical namespace)

```
azcopy copy 'C:\myDirectory' 'https://mystorageaccount.dfs.core.windows.net/mycontainer' --include-path
'photos;documents\myfile.txt' --recursive'
```

In this example, AzCopy transfers the `C:\myDirectory\photos` directory and the `C:\myDirectory\documents\myfile.txt` file. Include the `--recursive` option to transfer all files in the `C:\myDirectory\photos` directory.

You can also exclude files by using the `--exclude-path` option. To learn more, see [azcopy copy](#) reference docs.

## Use wildcard characters

Use the [azcopy copy](#) command with the `--include-pattern` option. Specify partial names that include the wildcard characters. Separate names by using a semicolon (`;`).

## Syntax

```
azcopy copy '<local-directory-path>' 'https://<storage-account-name>.<blob or
dfs>.core.windows.net/<container-name>' --include-pattern <:semicolon-separated-file-list-with-wildcard-
characters>
```

## Example

```
azcopy copy 'C:\myDirectory' 'https://mystorageaccount.blob.core.windows.net/mycontainer' --include-pattern
'myfile*.txt;*.pdf*'
```

## Example (hierarchical namespace)

```
azcopy copy 'C:\myDirectory' 'https://mystorageaccount.dfs.core.windows.net/mycontainer' --include-pattern
'myfile*.txt;*.pdf*'
```

You can also exclude files by using the `--exclude-pattern` option. To learn more, see [azcopy copy](#) reference docs.

The `--include-pattern` and `--exclude-pattern` options apply only to filenames and not to the path. If you want to copy all of the text files that exist in a directory tree, use the `-recursive` option to get the entire directory tree, and then use the `--include-pattern` and specify `*.txt` to get all of the text files.

## Upload files that were modified before or after a date and time

Use the [azcopy copy](#) command with the `--include-before` or `--include-after` option. Specify a date and time in ISO-8601 format (For example: `2020-08-19T15:04:00Z` ).

The following examples upload files that were modified on or after the specified date.

## Syntax

```
azcopy copy '<local-directory-path>*' 'https://<storage-account-name>.<blob or
dfs>.core.windows.net/<container-or-directory-name>' --include-after <Date-Time-in-ISO-8601-format>
```

## Example

```
azcopy copy 'C:\myDirectory*' 'https://mystorageaccount.blob.core.windows.net/mycontainer/FileDirectory' --include-after '2020-08-19T15:04:00Z'
```

## Example (hierarchical namespace)

```
azcopy copy 'C:\myDirectory*' 'https://mystorageaccount.dfs.core.windows.net/mycontainer/FileDirectory' --include-after '2020-08-19T15:04:00Z'
```

For detailed reference, see the [azcopy copy](#) reference docs.

## Upload with index tags

You can upload a file and add [blob index tags\(preview\)](#) to the target blob.

If you're using Azure AD authorization, your security principal must be assigned the [Storage Blob Data Owner](#) role or it must be given permission to the

`Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write` [Azure resource provider operation](#) via a custom Azure role. If you're using a Shared Access Signature (SAS) token, that token must provide access to the blob's tags via the `t` SAS permission.

To add tags, use the `--blob-tags` option along with a URL encoded key-value pair. For example, to add the key `my tag` and a value `my tag value`, you would add `--blob-tags='my%20tag=my%20tag%20value'` to the destination parameter.

Separate multiple index tags by using an ampersand (`&`). For example, if you want to add a key `my second tag` and a value `my second tag value`, the complete option string would be

```
--blob-tags='my%20tag=my%20tag%20value&my%20second%20tag=my%20second%20tag%20value'.
```

The following examples show how to use the `--blob-tags` option.

### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

## Upload a file

```
azcopy copy 'C:\myDirectory\myTextFile.txt'
'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile.txt' --blob-
tags='my%20tag=my%20tag%20value&my%20second%20tag=my%20second%20tag%20value'
```

## Upload a directory

```
azcopy copy 'C:\myDirectory' 'https://mystorageaccount.blob.core.windows.net/mycontainer' --recursive --
blob-tags='my%20tag=my%20tag%20value&my%20second%20tag=my%20second%20tag%20value'
```

## Upload directory contents

```
azcopy copy 'C:\myDirectory*' 'https://mystorageaccount.blob.core.windows.net/mycontainer/myBlobDirectory'
--blob-tags='my%20tag=my%20tag%20value&my%20second%20tag=my%20second%20tag%20value'
```

#### NOTE

If you specify a directory for the source, all the blobs that are copied to the destination will have the same tags that you specify in the command.

## Upload with optional flags

You can tweak your upload operation by using optional flags. Here's a few examples.

SCENARIO	FLAG
Upload files as Append Blobs or Page Blobs.	--blob-type=[BlockBlob PageBlob AppendBlob]
Upload to a specific access tier (such as the archive tier).	--block-blob-tier=[None Hot Cool Archive]

For a complete list, see [options](#).

## Next steps

Find more examples in these articles:

- [Examples: Download](#)
- [Examples: Copy between accounts](#)
- [Examples: Synchronize](#)
- [Examples: Amazon S3 buckets](#)
- [Examples: Google Cloud Storage](#)
- [Examples: Azure Files](#)
- [Tutorial: Migrate on-premises data to cloud storage by using AzCopy](#)

See these articles to configure settings, optimize performance, and troubleshoot issues:

- [AzCopy configuration settings](#)
- [Optimize the performance of AzCopy](#)
- [Find errors and resume jobs by using log and plan files in AzCopy](#)
- [Troubleshoot problems with AzCopy v10](#)

# Download blobs from Azure Blob Storage by using AzCopy

8/22/2022 • 5 minutes to read • [Edit Online](#)

You can download blobs and directories from Blob storage by using the AzCopy v10 command-line utility.

To see examples for other types of tasks such as uploading files, synchronizing with Blob storage, or copying blobs between accounts, see the links presented in the [Next Steps](#) section of this article.

## Get started

See the [Get started with AzCopy](#) article to download AzCopy and learn about the ways that you can provide authorization credentials to the storage service.

### NOTE

The examples in this article assume that you've provided authorization credentials by using Azure Active Directory (Azure AD).

If you'd rather use a SAS token to authorize access to blob data, then you can append that token to the resource URL in each AzCopy command. For example:

```
'https://<storage-account-name>.blob.core.windows.net/<container-name><SAS-token>'.
```

## Download a blob

Download a blob by using the [azcopy copy](#) command.

### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

### Syntax

```
azcopy copy 'https://<storage-account-name>.<blob or dfs>.core.windows.net/<container-name>/<blob-path>' '<local-file-path>'
```

### Example

```
azcopy copy 'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile.txt' 'C:\myDirectory\myTextFile.txt'
```

### Example (hierarchical namespace)

```
azcopy copy 'https://mystorageaccount.dfs.core.windows.net/mycontainer/myTextFile.txt' 'C:\myDirectory\myTextFile.txt'
```

#### NOTE

If the `Content-md5` property value of a blob contains a hash, AzCopy calculates an MD5 hash for downloaded data and verifies that the MD5 hash stored in the blob's `Content-md5` property matches the calculated hash. If these values don't match, the download fails unless you override this behavior by appending `--check-md5=NoCheck` or `--check-md5=LogOnly` to the copy command.

## Download a directory

Download a directory by using the [azcopy copy](#) command.

#### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

### Syntax

```
azcopy copy 'https://<storage-account-name>.<blob or dfs>.core.windows.net/<container-name>/<directory-path>' '<local-directory-path>' --recursive
```

### Example

```
azcopy copy 'https://mystorageaccount.blob.core.windows.net/mycontainer/myBlobDirectory' 'C:\myDirectory' --recursive
```

### Example (hierarchical namespace)

```
azcopy copy 'https://mystorageaccount.dfs.core.windows.net/mycontainer/myBlobDirectory' 'C:\myDirectory' --recursive
```

This example results in a directory named `C:\myDirectory\myBlobDirectory` that contains all of the downloaded blobs.

## Download directory contents

You can download the contents of a directory without copying the containing directory itself by using the wildcard symbol (\*).

#### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

#### NOTE

Currently, this scenario is supported only for accounts that don't have a hierarchical namespace.

### Syntax

```
azcopy copy 'https://<storage-account-name>.blob.core.windows.net/<container-name>/*' '<local-directory-path>'
```

## Example

```
azcopy copy 'https://mystorageaccount.blob.core.windows.net/mycontainer/myBlobDirectory/*' 'C:\myDirectory'
```

Append the `--recursive` flag to download files in all subdirectories.

## Download specific blobs

You can download specific blobs by using complete file names, partial names with wildcard characters (\*), or by using dates and times.

### TIP

These examples enclose path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

### Specify multiple complete blob names

Use the `azcopy copy` command with the `--include-path` option. Separate individual blob names by using a semicolon ( ; ).

## Syntax

```
azcopy copy 'https://<storage-account-name>.<blob or dfs>.core.windows.net/<container-or-directory-name>' '<local-directory-path>' --include-path <semicolon-separated-file-list>
```

## Example

```
azcopy copy 'https://mystorageaccount.blob.core.windows.net/mycontainer/FileDirectory' 'C:\myDirectory' --include-path 'photos;documents\myFile.txt' --recursive
```

### Example (hierarchical namespace)

```
azcopy copy 'https://mystorageaccount.dfs.core.windows.net/mycontainer/FileDirectory' 'C:\myDirectory' --include-path 'photos;documents\myFile.txt' --recursive
```

In this example, AzCopy transfers the

`https://mystorageaccount.blob.core.windows.net/mycontainer/FileDirectory/photos` directory and the  
`https://mystorageaccount.blob.core.windows.net/mycontainer/FileDirectory/documents/myFile.txt` file. Include the  
`--recursive` option to transfer all blobs in the  
`https://mystorageaccount.blob.core.windows.net/mycontainer/FileDirectory/photos` directory.

You can also exclude blobs by using the `--exclude-path` option. To learn more, see [azcopy copy](#) reference docs.

### Use wildcard characters

Use the `azcopy copy` command with the `--include-pattern` option. Specify partial names that include the wildcard characters. Separate names by using a semicolon ( ; ).

## Syntax

```
azcopy copy 'https://<storage-account-name>.<blob or dfs>.core.windows.net/<container-or-directory-name>' '<local-directory-path>' --include-pattern <semicolon-separated-file-list-with-wildcard-characters>
```

## Example

```
azcopy copy 'https://mystorageaccount.blob.core.windows.net/mycontainer/FileDirectory' 'C:\myDirectory' --include-pattern 'myFile*.txt;*.pdf'
```

## Example (hierarchical namespace)

```
azcopy copy 'https://mystorageaccount.dfs.core.windows.net/mycontainer/FileDirectory' 'C:\myDirectory' --include-pattern 'myFile*.txt;*.pdf'
```

You can also exclude blobs by using the `--exclude-pattern` option. To learn more, see [azcopy copy](#) reference docs.

The `--include-pattern` and `--exclude-pattern` options apply only to blob names and not to the path. If you want to copy all of the text files (blobs) that exist in a directory tree, use the `-recursive` option to get the entire directory tree, and then use the `-include-pattern` and specify `*.txt` to get all of the text files.

### Download blobs that were modified before or after a date and time

Use the [azcopy copy](#) command with the `--include-before` or `--include-after` option. Specify a date and time in ISO-8601 format (For example: `2020-08-19T15:04:00Z`).

The following examples download files that were modified on or after the specified date.

## Syntax

```
azcopy copy 'https://<storage-account-name>.<blob or dfs>.core.windows.net/<container-or-directory-name>/*' '<local-directory-path>' --include-after <Date-Time-in-ISO-8601-format>
```

## Example

```
azcopy copy 'https://mystorageaccount.blob.core.windows.net/mycontainer/FileDirectory/*' 'C:\myDirectory' --include-after '2020-08-19T15:04:00Z'
```

## Example (hierarchical namespace)

```
azcopy copy 'https://mystorageaccount.dfs.core.windows.net/mycontainer/FileDirectory/*' 'C:\myDirectory' --include-after '2020-08-19T15:04:00Z'
```

For detailed reference, see the [azcopy copy](#) reference docs.

### Download previous versions of a blob

If you've enabled [blob versioning](#), you can download one or more previous versions of a blob.

First, create a text file that contains a list of [version IDs](#). Each version ID must appear on a separate line. For example:

```
2020-08-17T05:50:34.2199403Z
2020-08-17T05:50:34.5041365Z
2020-08-17T05:50:36.7607103Z
```

Then, use the [azcopy copy](#) command with the `--list-of-versions` option. Specify the location of the text file that contains the list of versions (For example: `D:\\list-of-versions.txt`).

### Download a blob snapshot

You can download a [blob snapshot](#) by referencing the [DateTime](#) value of a blob snapshot.

## Syntax

```
azcopy copy 'https://<storage-account-name>.<blob or dfs>.core.windows.net/<container-name>/<blob-path>?sharesnapshot=<DateTime-of-snapshot>' '<local-file-path>'
```

## Example

```
azcopy copy 'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile.txt?sharesnapshot=2020-09-23T08:21:07.000000Z' 'C:\myDirectory\myTextFile.txt'
```

## Example (hierarchical namespace)

```
azcopy copy 'https://mystorageaccount.dfs.core.windows.net/mycontainer/myTextFile.txt?sharesnapshot=2020-09-23T08:21:07.000000Z' 'C:\myDirectory\myTextFile.txt'
```

### NOTE

If you are using a SAS token to authorize access to blob data, then append snapshot **DateTime** after the SAS token. For example:

```
'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile.txt?sv=2018-03-28&ss=bjqs&srs=sco&sp=rjk1hjup&se=2019-05-10T04:37:48Z&st=2019-05-09T20:37:48Z&spr=https&sig=%2F5OVEFFsKDqRry4bk3qz1vAQFWY5DDzp2%2B%2F3Eykf%2FJLs%3D&sharesnapshot=2020-09-23T08:21:07.000000Z'
```

## Download with optional flags

You can tweak your download operation by using optional flags. Here's a few examples.

SCENARIO	FLAG
Automatically decompress files.	--decompress
Specify how detailed you want your copy-related log entries to be.	--log-level=[WARNING ERROR INFO NONE]
Specify if and how to overwrite the conflicting files and blobs at the destination.	--overwrite=[true false ifSourceNewer prompt]

For a complete list, see [options](#).

## Next steps

Find more examples in these articles:

- [Examples: Upload](#)
- [Examples: Copy between account](#)
- [Examples: Synchronize](#)
- [Examples: Amazon S3 buckets](#)
- [Examples: Google Cloud Storage](#)
- [Examples: Azure Files](#)
- [Tutorial: Migrate on-premises data to cloud storage by using AzCopy](#)

See these articles to configure settings, optimize performance, and troubleshoot issues:

- [AzCopy configuration settings](#)
- [Optimize the performance of AzCopy](#)

- Find errors and resume jobs by using log and plan files in AzCopy
- Troubleshoot problems with AzCopy v10

# Copy blobs between Azure storage accounts by using AzCopy

8/22/2022 • 5 minutes to read • [Edit Online](#)

You can copy blobs, directories, and containers between storage accounts by using the AzCopy v10 command-line utility.

To see examples for other types of tasks such as uploading files, downloading blobs, and synchronizing with Blob storage, see the links presented in the [Next Steps](#) section of this article.

AzCopy uses [server-to-server APIs](#), so data is copied directly between storage servers. These copy operations don't use the network bandwidth of your computer.

## Get started

See the [Get started with AzCopy](#) article to download AzCopy and learn about the ways that you can provide authorization credentials to the storage service.

### NOTE

The examples in this article assume that you've provided authorization credentials by using Azure Active Directory (Azure AD) and that your Azure AD identity has the proper role assignments for both source and destination accounts.

Alternatively, you can append a SAS token to either the source or destination URL in each AzCopy command. For example:

```
azcopy copy 'https://<source-storage-account-name>.blob.core.windows.net/<container-name>/<blob-path><SAS-token>' 'https://<destination-storage-account-name>.blob.core.windows.net/<container-name>/<blob-path><SAS-token>'
```

## Guidelines

Apply the following guidelines to your AzCopy commands.

- Your client must have network access to both the source and destination storage accounts. To learn how to configure the network settings for each storage account, see [Configure Azure Storage firewalls and virtual networks](#).
- If you copy to a premium block blob storage account, omit the access tier of a blob from the copy operation by setting the `s2s-preserve-access-tier` to `false` (For example: `--s2s-preserve-access-tier=false`). Premium block blob storage accounts don't support access tiers.
- If you copy to or from an account that has a hierarchical namespace, use `blob.core.windows.net` instead of `dfs.core.windows.net` in the URL syntax. [Multi-protocol access on Data Lake Storage](#) enables you to use `blob.core.windows.net`, and it is the only supported syntax for account to account copy scenarios.
- You can increase the throughput of copy operations by setting the value of the `AZCOPY_CONCURRENCY_VALUE` environment variable. To learn more, see [Increase Concurrency](#).
- If the source blobs have index tags, and you want to retain those tags, you'll have to reapply them to the destination blobs. For information about how to set index tags, see the [Copy blobs to another storage account with index tags](#) section of this article.

## Copy a blob

Copy a blob to another storage account by using the [azcopy copy](#) command.

### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

### Syntax

```
azcopy copy 'https://<source-storage-account-name>.blob.core.windows.net/<container-name>/<blob-path>'
'https://<destination-storage-account-name>.blob.core.windows.net/<container-name>/<blob-path>'
```

### Example

```
azcopy copy 'https://mysourceaccount.blob.core.windows.net/mycontainer/myTextFile.txt'
'https://mydestinationaccount.blob.core.windows.net/mycontainer/myTextFile.txt'
```

The copy operation is synchronous so when the command returns, that indicates that all files have been copied.

## Copy a directory

Copy a directory to another storage account by using the [azcopy copy](#) command.

### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

### Syntax

```
azcopy copy 'https://<source-storage-account-name>.blob.core.windows.net/<container-name>/<directory-path>'
'https://<destination-storage-account-name>.blob.core.windows.net/<container-name>' --recursive
```

### Example

```
azcopy copy 'https://mysourceaccount.blob.core.windows.net/mycontainer/myBlobDirectory'
'https://mydestinationaccount.blob.core.windows.net/mycontainer' --recursive
```

The copy operation is synchronous so when the command returns, that indicates that all files have been copied.

## Copy a container

Copy a container to another storage account by using the [azcopy copy](#) command.

### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

### Syntax

```
azcopy copy 'https://<source-storage-account-name>.blob.core.windows.net/<container-name>'
'https://<destination-storage-account-name>.blob.core.windows.net/<container-name>' --recursive
```

## Example

```
azcopy copy 'https://mysourceaccount.blob.core.windows.net/mycontainer'
'https://mydestinationaccount.blob.core.windows.net/mycontainer' --recursive
```

The copy operation is synchronous so when the command returns, that indicates that all files have been copied.

## Copy containers, directories, and blobs

Copy all containers, directories, and blobs to another storage account by using the [azcopy copy](#) command.

### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

## Syntax

```
azcopy copy 'https://<source-storage-account-name>.blob.core.windows.net/' 'https://<destination-storage-
account-name>.blob.core.windows.net/' --recursive
```

## Example

```
azcopy copy 'https://mysourceaccount.blob.core.windows.net/'
'https://mydestinationaccount.blob.core.windows.net' --recursive
```

The copy operation is synchronous so when the command returns, that indicates that all files have been copied.

## Copy blobs and add index tags

Copy blobs to another storage account and add [blob index tags\(preview\)](#) to the target blob.

If you're using Azure AD authorization, your security principal must be assigned the [Storage Blob Data Owner](#) role or it must be given permission to the

`Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write` [Azure resource provider operation](#) via a custom Azure role. If you're using a Shared Access Signature (SAS) token, that token must provide access to the blob's tags via the `t` SAS permission.

To add tags, use the `--blob-tags` option along with a URL encoded key-value pair.

For example, to add the key `my tag` and a value `my tag value`, you would add

```
--blob-tags='my%20tag=my%20tag%20value'
```

Separate multiple index tags by using an ampersand (&). For example, if you want to add a key `my second tag` and a value `my second tag value`, the complete option string would be

```
--blob-tags='my%20tag=my%20tag%20value&my%20second%20tag=my%20second%20tag%20value'.
```

The following examples show how to use the `--blob-tags` option.

**TIP**

These examples enclose path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

## Blob example

```
azcopy copy 'https://mysourceaccount.blob.core.windows.net/mycontainer/myTextFile.txt'
'https://mydestinationaccount.blob.core.windows.net/mycontainer/myTextFile.txt' --blob-
tags='my%20tag=my%20tag%20value&my%20second%20tag=my%20second%20tag%20value'
```

## Directory example

```
azcopy copy 'https://mysourceaccount.blob.core.windows.net/mycontainer/myBlobDirectory'
'https://mydestinationaccount.blob.core.windows.net/mycontainer' --recursive --blob-
tags='my%20tag=my%20tag%20value&my%20second%20tag=my%20second%20tag%20value'
```

## Container example

```
azcopy copy 'https://mysourceaccount.blob.core.windows.net/mycontainer'
'https://mydestinationaccount.blob.core.windows.net/mycontainer' --recursive --blob-
tags='my%20tag=my%20tag%20value&my%20second%20tag=my%20second%20tag%20value'
```

## Account example

```
azcopy copy 'https://mysourceaccount.blob.core.windows.net/'
'https://mydestinationaccount.blob.core.windows.net' --recursive --blob-
tags='my%20tag=my%20tag%20value&my%20second%20tag=my%20second%20tag%20value'
```

The copy operation is synchronous so when the command returns, that indicates that all files have been copied.

**NOTE**

If you specify a directory, container, or account for the source, all the blobs that are copied to the destination will have the same tags that you specify in the command.

## Copy with optional flags

You can tweak your copy operation by using optional flags. Here's a few examples.

SCENARIO	FLAG
Copy blobs as Block, Page, or Append Blobs.	--blob-type=[BlockBlob PageBlob AppendBlob]
Copy to a specific access tier (such as the archive tier).	--block-blob-tier=[None Hot Cool Archive]
Automatically decompress files.	--decompress=[gzip deflate]

For a complete list, see [options](#).

## Next steps

Find more examples in these articles:

- [Examples: Upload](#)
- [Examples: Download](#)
- [Examples: Synchronize](#)
- [Examples: Amazon S3 buckets](#)
- [Examples: Google Cloud Storage](#)
- [Examples: Azure Files](#)
- [Tutorial: Migrate on-premises data to cloud storage by using AzCopy](#)

See these articles to configure settings, optimize performance, and troubleshoot issues:

- [AzCopy configuration settings](#)
- [Optimize the performance of AzCopy](#)
- [Find errors and resume jobs by using log and plan files in AzCopy](#)
- [Troubleshoot problems with AzCopy v10](#)

# Synchronize with Azure Blob storage by using AzCopy

8/22/2022 • 5 minutes to read • [Edit Online](#)

You can synchronize local storage with Azure Blob storage by using the AzCopy v10 command-line utility.

You can synchronize the contents of a local file system with a blob container. You can also synchronize containers and virtual directories with one another. Synchronization is one way. In other words, you choose which of these two endpoints is the source and which one is the destination. Synchronization also uses server to server APIs. The examples presented in this section also work with accounts that have a hierarchical namespace.

## NOTE

The current release of AzCopy doesn't synchronize between other sources and destinations (For example: File storage or Amazon Web Services (AWS) S3 buckets).

To see examples for other types of tasks such as uploading files, downloading blobs, or copying blobs between accounts, see the links presented in the [Next Steps](#) section of this article.

## Get started

See the [Get started with AzCopy](#) article to download AzCopy and learn about the ways that you can provide authorization credentials to the storage service.

## NOTE

The examples in this article assume that you've provided authorization credentials by using Azure Active Directory (Azure AD).

If you'd rather use a SAS token to authorize access to blob data, then you can append that token to the resource URL in each AzCopy command. For example:

```
'https://<storage-account-name>.blob.core.windows.net/<container-name><SAS-token>' .
```

## Guidelines

- The `sync` command compares file names and last modified timestamps. Set the `--delete-destination` optional flag to a value of `true` or `prompt` to delete files in the destination directory if those files no longer exist in the source directory.
- If you set the `--delete-destination` flag to `true`, AzCopy deletes files without providing a prompt. If you want a prompt to appear before AzCopy deletes a file, set the `--delete-destination` flag to `prompt`.
- If you plan to set the `--delete-destination` flag to `prompt` or `false`, consider using the `copy` command instead of the `sync` command and set the `--overwrite` parameter to `ifSourceNewer`. The `copy` command consumes less memory and incurs less billing costs because a copy operation doesn't have to index the source or destination prior to moving files.
- To prevent accidental deletions, make sure to enable the `soft delete` feature before you use the `--delete-destination=prompt|true` flag.

- The machine on which you run the sync command should have an accurate system clock because the last modified times are critical in determining whether a file should be transferred. If your system has significant clock skew, avoid modifying files at the destination too close to the time that you plan to run a sync command.

## Update a container with changes to a local file system

In this case, the container is the destination, and the local file system is the source.

### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

### Syntax

```
azcopy sync '<local-directory-path>' 'https://<storage-account-name>.blob.core.windows.net/<container-name>' --recursive
```

### Example

```
azcopy sync 'C:\myDirectory' 'https://mystorageaccount.blob.core.windows.net/mycontainer' --recursive
```

## Update a local file system with changes to a container

In this case, the local file system is the destination, and the container is the source.

### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

### Syntax

```
azcopy sync 'https://<storage-account-name>.blob.core.windows.net/<container-name>' 'C:\myDirectory' --recursive
```

### Example

```
azcopy sync 'https://mystorageaccount.blob.core.windows.net/mycontainer' 'C:\myDirectory' --recursive
```

## Update a container with changes in another container

The first container that appears in this command is the source. The second one is the destination. Make sure to append a SAS token to each source URL.

If you provide authorization credentials by using Azure Active Directory (Azure AD), you can omit the SAS token only from the destination URL. Make sure that you've set up the proper roles in your destination account. See [Option 1: Use Azure Active Directory](#).

**TIP**

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

## Syntax

```
azcopy sync 'https://<source-storage-account-name>.blob.core.windows.net/<container-name>/<SAS-token>'
'https://<destination-storage-account-name>.blob.core.windows.net/<container-name>' --recursive
```

## Example

```
azcopy sync 'https://mysourceaccount.blob.core.windows.net/mycontainer?sv=2018-03-
28&ss=bfqt&srt=sco&sp=rwdlacup&se=2019-07-04T05:30:08Z&st=2019-07-
03T21:30:08Z&spr=https&sig=CAFhgnc9gdGktvB=ska7bAiqIdM845yiyFwdMH481QA8%3D'
'https://mydestinationaccount.blob.core.windows.net/mycontainer' --recursive
```

## Update a directory with changes to a directory in another container

The first directory that appears in this command is the source. The second one is the destination. Make sure to append a SAS token to each source URL.

If you provide authorization credentials by using Azure Active Directory (Azure AD), you can omit the SAS token only from the destination URL. Make sure that you've set up the proper roles in your destination account. See [Option 1: Use Azure Active Directory](#).

**TIP**

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

## Syntax

```
azcopy sync 'https://<source-storage-account-name>.blob.core.windows.net/<container-name>/<directory-
name>/<SAS-token>' 'https://<destination-storage-account-name>.blob.core.windows.net/<container-
name>/<directory-name>' --recursive
```

## Example

```
azcopy sync 'https://mysourceaccount.blob.core.windows.net/<container-name>/myDirectory?sv=2018-03-
28&ss=bfqt&srt=sco&sp=rwdlacup&se=2019-07-04T05:30:08Z&st=2019-07-
03T21:30:08Z&spr=https&sig=CAFhgnc9gdGktvB=ska7bAiqIdM845yiyFwdMH481QA8%3D'
'https://mydestinationaccount.blob.core.windows.net/mycontainer/myDirectory' --recursive
```

## Synchronize with optional flags

You can tweak your sync operation by using optional flags. Here's a few examples.

SCENARIO	FLAG
Specify how strictly MD5 hashes should be validated when downloading.	--check-md5= [NoCheck LogOnly FailIfDifferent FailIfDifferentOrMissing]

SCENARIO	FLAG
Exclude files based on a pattern.	--exclude-path
Specify how detailed you want your sync-related log entries to be.	--log-level=[WARNING ERROR INFO NONE]

For a complete list of flags, see [options](#).

#### NOTE

The `--recursive` flag is set to `true` by default. The `--exclude-pattern` and `--include-pattern` flags apply to only file names and not other parts of the file path.

## Next steps

Find more examples in these articles:

- [Examples: Upload](#)
- [Examples: Download](#)
- [Examples: Copy between accounts](#)
- [Examples: Amazon S3 buckets](#)
- [Examples: Google Cloud Storage](#)
- [Examples: Azure Files](#)
- [Tutorial: Migrate on-premises data to cloud storage by using AzCopy](#)

See these articles to configure settings, optimize performance, and troubleshoot issues:

- [AzCopy configuration settings](#)
- [Optimize the performance of AzCopy](#)
- [Find errors and resume jobs by using log and plan files in AzCopy](#)
- [Troubleshoot problems with AzCopy v10](#)

# Replace blob properties and metadata by using AzCopy v10 (preview)

8/22/2022 • 3 minutes to read • [Edit Online](#)

You can use AzCopy to change the [access tier](#) of one or more blobs and replace (*overwrite*) the metadata, and index tags of one or more blobs.

## IMPORTANT

This capability is currently in PREVIEW. See the [Supplemental Terms of Use for Microsoft Azure Previews](#) for legal terms that apply to Azure features that are in beta, preview, or otherwise not yet released into general availability.

## Get started

See the [Get started with AzCopy](#) article to download AzCopy and learn about the ways that you can provide authorization credentials to the storage service.

## NOTE

The examples in this article assume that you've provided authorization credentials by using Azure Active Directory (Azure AD).

If you'd rather use a SAS token to authorize access to blob data, then you can append that token to the resource URL in each AzCopy command. For example:

```
'https://<storage-account-name>.blob.core.windows.net/<container-name><SAS-token>' .
```

## Change the access tier

To change the access tier of a blob, use the [azcopy set-properties](#) command and set the `-block-blob-tier` parameter to the name of the access tier.

## TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

## Syntax

```
azcopy set-properties 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<blob-name>' --block-blob-tier=<access-tier>
```

## Example

```
azcopy set-properties 'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile.txt' --block-blob-tier=hot
```

To change the access tier for all blobs in a virtual directory, refer to the virtual directory name instead of the

blob name, and then append `--recursive=true` to the command.

## Example

```
azcopy set-properties 'https://mystorageaccount.blob.core.windows.net/mycontainer/myvirtualdirectory' --block-blob-tier=hot --recursive=true
```

To *rehydrate* a blob from the archive tier to an online tier, set the `--rehydrate-priority` to `standard` or `high`. By default, this parameter is set to `standard`. To learn more about the trade offs of each option, see [Rehydration priority](#).

## Example

```
azcopy set-properties 'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile.txt' --block-blob-tier=hot --rehydrate-priority=high
```

# Replace metadata

To replace the metadata of a blob, use the [azcopy set-properties](#) command and set the `--metadata` parameter to one or more key-value pairs.

### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

## Syntax

```
azcopy set-properties 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<blob-name>' --metadata=<key>=<value>;<key>=<value>
```

## Example

```
azcopy set-properties 'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile.txt' --metadata=mykey1=myvalue1;mykey2=myvalue2
```

To replace the metadata for all blobs in a virtual directory, refer to the virtual directory name instead of the blob name, and then append `--recursive=true` to the command.

## Example

```
azcopy set-properties 'https://mystorageaccount.blob.core.windows.net/mycontainer/myvirtualdirectory' --metadata=mykey1=myvalue1;mykey2=myvalue2 --recursive=true
```

To clear metadata, omit the tags and append `--metadata=clear` to the end of the command.

## Example

```
azcopy set-properties 'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile.txt' --metadata=clear
```

# Replace index tags

To replace the index tags of a blob, use the [azcopy set-properties](#) command and set the `--blob-tags` parameter to one or more key-value pairs. Setting blob index tags can be performed by the [Storage Blob Data Owner](#) and by anyone with a Shared Access Signature that has permission to access the blob's tags (the `t` SAS permission). In addition, RBAC users with the `Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write` permission can perform this operation.

#### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

## Syntax

```
azcopy set-properties 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<blob-name>' --blob-tags=<tag>=<value>;<tag>=<value>
```

## Example

```
azcopy set-properties 'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile.txt' --blob-tags=mytag1=mytag1value;mytag2=mytag2value
```

To replace the index tags for all blobs in a virtual directory, refer to the virtual directory name instead of the blob name, and then append `--recursive=true` to the command.

## Example

```
azcopy set-properties 'https://mystorageaccount.blob.core.windows.net/mycontainer/myvirtualdirectory' --blob-tags=mytag1=mytag1value;mytag2=mytag2value
```

## Next steps

Find more examples in these articles:

- [Examples: Upload](#)
- [Examples: Download](#)
- [Examples: Copy between accounts](#)
- [Examples: Synchronize](#)
- [Examples: Amazon S3 buckets](#)
- [Examples: Google Cloud Storage](#)
- [Examples: Azure Files](#)
- [Tutorial: Migrate on-premises data to cloud storage by using AzCopy](#)

See these articles to configure settings, optimize performance, and troubleshoot issues:

- [AzCopy configuration settings](#)
- [Optimize the performance of AzCopy](#)
- [Find errors and resume jobs by using log and plan files in AzCopy](#)
- [Troubleshoot problems with AzCopy v10](#)

# Copy data from Amazon S3 to Azure Storage by using AzCopy

8/22/2022 • 5 minutes to read • [Edit Online](#)

AzCopy is a command-line utility that you can use to copy blobs or files to or from a storage account. This article helps you copy objects, directories, and buckets from Amazon Web Services (AWS) S3 to Azure Blob Storage by using AzCopy.

## Choose how you'll provide authorization credentials

- To authorize with the Azure Storage, use Azure Active Directory (AD) or a Shared Access Signature (SAS) token.
- To authorize with AWS S3, use an AWS access key and a secret access key.

### Authorize with Azure Storage

See the [Get started with AzCopy](#) article to download AzCopy, and choose how you'll provide authorization credentials to the storage service.

#### NOTE

The examples in this article assume that you've authenticated your identity by using the `AzCopy login` command. AzCopy then uses your Azure AD account to authorize access to data in Blob storage.

If you'd rather use a SAS token to authorize access to blob data, then you can append that token to the resource URL in each AzCopy command.

For example: `https://mystorageaccount.blob.core.windows.net/mycontainer?<SAS-token>`.

### Authorize with AWS S3

Gather your AWS access key and secret access key, and then set these environment variables:

OPERATING SYSTEM	COMMAND
Windows	<code>set AWS_ACCESS_KEY_ID=&lt;access-key&gt;</code> <code>set AWS_SECRET_ACCESS_KEY=&lt;secret-access-key&gt;</code>
Linux	<code>export AWS_ACCESS_KEY_ID=&lt;access-key&gt;</code> <code>export AWS_SECRET_ACCESS_KEY=&lt;secret-access-key&gt;</code>
macOS	<code>export AWS_ACCESS_KEY_ID=&lt;access-key&gt;</code> <code>export AWS_SECRET_ACCESS_KEY=&lt;secret-access-key&gt;</code>

## Copy objects, directories, and buckets

AzCopy uses the [Put Block From URL](#) API, so data is copied directly between AWS S3 and storage servers. These copy operations don't use the network bandwidth of your computer.

## TIP

The examples in this section enclose path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

These examples also work with accounts that have a hierarchical namespace. [Multi-protocol access on Data Lake Storage](#) enables you to use the same URL syntax (`blob.core.windows.net`) on those accounts.

## Copy an object

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

### Syntax

```
azcopy copy 'https://s3.amazonaws.com/<bucket-name>/<object-name>' 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<blob-name>'
```

### Example

```
azcopy copy 'https://s3.amazonaws.com/mybucket/myobject'
'https://mystorageaccount.blob.core.windows.net/mycontainer/myblob'
```

## NOTE

Examples in this article use path-style URLs for AWS S3 buckets (For example:

`http://s3.amazonaws.com/<bucket-name>` ).

You can also use virtual hosted-style URLs as well (For example: `http://bucket.s3.amazonaws.com` ).

To learn more about virtual hosting of buckets, see [Virtual Hosting of Buckets](#).

## Copy a directory

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

### Syntax

```
azcopy copy 'https://s3.amazonaws.com/<bucket-name>/<directory-name>' 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<directory-name>' --recursive=true
```

### Example

```
azcopy copy 'https://s3.amazonaws.com/mybucket/mydirectory'
'https://mystorageaccount.blob.core.windows.net/mycontainer/mydirectory' --recursive=true
```

## NOTE

This example appends the `--recursive` flag to copy files in all sub-directories.

## Copy the contents of a directory

You can copy the contents of a directory without copying the containing directory itself by using the wildcard symbol (\*).

### Syntax

```
azcopy copy 'https://s3.amazonaws.com/<bucket-name>/<directory-name>/*' 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<directory-name>' --recursive=true
```

## Example

```
azcopy copy 'https://s3.amazonaws.com/mybucket/mydirectory/*'
'https://mystorageaccount.blob.core.windows.net/mycontainer/mydirectory' --recursive=true
```

### Copy a bucket

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

#### Syntax

```
azcopy copy 'https://s3.amazonaws.com/<bucket-name>' 'https://<storage-account-name>.blob.core.windows.net/<container-name>' --recursive=true
```

### Example

```
azcopy copy 'https://s3.amazonaws.com/mybucket' 'https://mystorageaccount.blob.core.windows.net/mycontainer'
--recursive=true
```

### Copy all buckets in all regions

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

#### Syntax

```
azcopy copy 'https://s3.amazonaws.com/' 'https://<storage-account-name>.blob.core.windows.net' --
recursive=true
```

### Example

```
azcopy copy 'https://s3.amazonaws.com' 'https://mystorageaccount.blob.core.windows.net' --recursive=true
```

### Copy all buckets in a specific S3 region

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

#### Syntax

```
azcopy copy 'https://s3-<region-name>.amazonaws.com/' 'https://<storage-account-name>.blob.core.windows.net'
--recursive=true
```

### Example

```
azcopy copy 'https://s3-rds.eu-north-1.amazonaws.com' 'https://mystorageaccount.blob.core.windows.net' --
recursive=true
```

## Handle differences in object naming rules

AWS S3 has a different set of naming conventions for bucket names as compared to Azure blob containers. You can read about them [here](#). If you choose to copy a group of buckets to an Azure storage account, the copy operation might fail because of naming differences.

AzCopy handles two of the most common issues that can arise; buckets that contain periods and buckets that contain consecutive hyphens. AWS S3 bucket names can contain periods and consecutive hyphens, but a container in Azure can't. AzCopy replaces periods with hyphens and consecutive hyphens with a number that represents the number of consecutive hyphens (For example: a bucket named `my----bucket` becomes `my-4-bucket`).

Also, as AzCopy copies over files, it checks for naming collisions and attempts to resolve them. For example, if

there are buckets with the name `bucket-name` and `bucket.name`, AzCopy resolves a bucket named `bucket.name` first to `bucket-name` and then to `bucket-name-2`.

## Handle differences in object metadata

AWS S3 and Azure allow different sets of characters in the names of object keys. You can read about the characters that AWS S3 uses [here](#). On the Azure side, blob object keys adhere to the naming rules for [C# identifiers](#).

As part of an AzCopy `copy` command, you can provide a value for optional the `s2s-handle-invalid-metadata` flag that specifies how you would like to handle files where the metadata of the file contains incompatible key names. The following table describes each flag value.

FLAG VALUE	DESCRIPTION
<code>ExcludeIfInvalid</code>	(Default option) The metadata isn't included in the transferred object. AzCopy logs a warning.
<code>FailIfInvalid</code>	Objects aren't copied. AzCopy logs an error and includes that error in the failed count that appears in the transfer summary.
<code>RenameIfInvalid</code>	AzCopy resolves the invalid metadata key, and copies the object to Azure using the resolved metadata key value pair. To learn exactly what steps AzCopy takes to rename object keys, see the <a href="#">How AzCopy renames object keys</a> section below. If AzCopy is unable to rename the key, then the object won't be copied.

### How AzCopy renames object keys

AzCopy performs these steps:

1. Replaces invalid characters with '\_'.  
2. Adds the string `rename_` to the beginning of a new valid key.

This key will be used to save the original metadata **value**.

3. Adds the string `rename_key_` to the beginning of a new valid key. This key will be used to save original metadata **invalid key**. You can use this key to try to recover the metadata in Azure side since metadata key is preserved as a value on the Blob storage service.

## Next steps

Find more examples in these articles:

- [Examples: Upload](#)
- [Examples: Download](#)
- [Examples: Copy between accounts](#)
- [Examples: Synchronize](#)
- [Examples: Google Cloud Storage](#)
- [Examples: Azure Files](#)
- [Tutorial: Migrate on-premises data to cloud storage by using AzCopy](#)

See these articles to configure settings, optimize performance, and troubleshoot issues:

- AzCopy configuration settings
- Optimize the performance of AzCopy
- Troubleshoot AzCopy V10 issues in Azure Storage by using log files

# Copy data from Google Cloud Storage to Azure Storage by using AzCopy

8/22/2022 • 5 minutes to read • [Edit Online](#)

AzCopy is a command-line utility that you can use to copy blobs or files to or from a storage account. This article helps you copy objects, directories, and buckets from Google Cloud Storage to Azure Blob Storage by using AzCopy.

## Choose how you'll provide authorization credentials

- To authorize with Azure Storage, use Azure Active Directory (AD) or a Shared Access Signature (SAS) token.
- To authorize with Google Cloud Storage, use a service account key.

### Authorize with Azure Storage

See the [Get started with AzCopy](#) article to download AzCopy and learn about the ways that you can provide authorization credentials to the storage service.

#### NOTE

The examples in this article assume that you've provided authorization credentials by using Azure Active Directory (Azure AD).

If you'd rather use a SAS token to authorize access to blob data, then you can append that token to the resource URL in each AzCopy command. For example:

```
'https://<storage-account-name>.blob.core.windows.net/<container-name><SAS-token>' .
```

### Authorize with Google Cloud Storage

To authorize with Google Cloud Storage, you'll use a service account key. For information about how to create a service account key, see [Creating and managing service account keys](#).

After you've obtained a service key, set the `GOOGLE_APPLICATION_CREDENTIALS` environment variable to absolute path to the service account key file:

OPERATING SYSTEM	COMMAND
Windows	<pre>set GOOGLE_APPLICATION_CREDENTIALS=&lt;path-to-service-account-key&gt;</pre>
Linux	<pre>export GOOGLE_APPLICATION_CREDENTIALS=&lt;path-to-service-account-key&gt;</pre>
macOS	<pre>export GOOGLE_APPLICATION_CREDENTIALS=&lt;path-to-service-account-key&gt;</pre>

## Copy objects, directories, and buckets

AzCopy uses the [Put Block From URL](#) API, so data is copied directly between Google Cloud Storage and storage servers. These copy operations don't use the network bandwidth of your computer.

## TIP

The examples in this section enclose path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

These examples also work with accounts that have a hierarchical namespace. [Multi-protocol access on Data Lake Storage](#) enables you to use the same URL syntax (`blob.core.windows.net`) on those accounts.

## Copy an object

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

### Syntax

```
azcopy copy 'https://storage.cloud.google.com/<bucket-name>/<object-name>' 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<blob-name>'
```

### Example

```
azcopy copy 'https://storage.cloud.google.com/mybucket/myobject'
'https://mystorageaccount.blob.core.windows.net/mycontainer/myblob'
```

## Copy a directory

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

### Syntax

```
azcopy copy 'https://storage.cloud.google.com/<bucket-name>/<directory-name>' 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<directory-name>' --recursive=true
```

### Example

```
azcopy copy 'https://storage.cloud.google.com/mybucket/mydirectory'
'https://mystorageaccount.blob.core.windows.net/mycontainer/mydirectory' --recursive=true
```

## NOTE

This example appends the `--recursive` flag to copy files in all sub-directories.

## Copy the contents of a directory

You can copy the contents of a directory without copying the containing directory itself by using the wildcard symbol (\*).

### Syntax

```
azcopy copy 'https://storage.cloud.google.com/<bucket-name>/<directory-name>/*' 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<directory-name>' --recursive=true
```

### Example

```
azcopy copy 'https://storage.cloud.google.com/mybucket/mydirectory/*'
'https://mystorageaccount.blob.core.windows.net/mycontainer/mydirectory' --recursive=true
```

## Copy a Cloud Storage bucket

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

## Syntax

```
azcopy copy 'https://storage.cloud.google.com/<bucket-name>' 'https://<storage-account-name>.blob.core.windows.net' --recursive=true
```

## Example

```
azcopy copy 'https://storage.cloud.google.com/mybucket' 'https://mystorageaccount.blob.core.windows.net' --recursive=true
```

## Copy all buckets in a Google Cloud project

First, set the `GOOGLE_CLOUD_PROJECT` to project ID of Google Cloud project.

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

## Syntax

```
azcopy copy 'https://storage.cloud.google.com/' 'https://<storage-account-name>.blob.core.windows.net' --recursive=true
```

## Example

```
azcopy copy 'https://storage.cloud.google.com/' 'https://mystorageaccount.blob.core.windows.net' --recursive=true
```

## Copy a subset of buckets in a Google Cloud project

First, set the `GOOGLE_CLOUD_PROJECT` to project ID of Google Cloud project.

Copy a subset of buckets by using a wildcard symbol (\*) in the bucket name. Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

## Syntax

```
azcopy copy 'https://storage.cloud.google.com/<bucket*name>' 'https://<storage-account-name>.blob.core.windows.net' --recursive=true
```

## Example

```
azcopy copy 'https://storage.cloud.google.com/my*bucket' 'https://mystorageaccount.blob.core.windows.net' --recursive=true
```

## Handle differences in bucket naming rules

Google Cloud Storage has a different set of naming conventions for bucket names as compared to Azure blob containers. You can read about them [here](#). If you choose to copy a group of buckets to an Azure storage account, the copy operation might fail because of naming differences.

AzCopy handles three of the most common issues that can arise; buckets that contain periods, buckets that contain consecutive hyphens, and buckets that contain underscores. Google Cloud Storage bucket names can contain periods and consecutive hyphens, but a container in Azure can't. AzCopy replaces periods with hyphens and consecutive hyphens with a number that represents the number of consecutive hyphens (For example: a bucket named `my---bucket` becomes `my-4-bucket` . If the bucket name has an underscore (`_`), then AzCopy replaces the underscore with a hyphen. For example, a bucket named `my_bucket` becomes `my-bucket` .

## Handle differences in object naming rules

Google Cloud Storage has a different set of naming conventions for object names as compared to Azure blobs.

You can read about them [here](#).

Azure Storage does not permit object names (or any segment in the virtual directory path) to end with trailing dots (For example `my-bucket...`). Trailing dots are trimmed off when the copy operation is performed.

## Handle differences in object metadata

Google Cloud Storage and Azure allow different sets of characters in the names of object keys. You can read about metadata in Google Cloud Storage [here](#). On the Azure side, blob object keys adhere to the naming rules for [C# identifiers](#).

As part of an AzCopy `copy` command, you can provide a value for optional the `s2s-handle-invalid-metadata` flag that specifies how you would like to handle files where the metadata of the file contains incompatible key names. The following table describes each flag value.

FLAG VALUE	DESCRIPTION
<code>ExcludeIfInvalid</code>	(Default option) The metadata isn't included in the transferred object. AzCopy logs a warning.
<code>FailIfInvalid</code>	Objects aren't copied. AzCopy logs an error and includes that error in the failed count that appears in the transfer summary.
<code>RenameIfInvalid</code>	AzCopy resolves the invalid metadata key, and copies the object to Azure using the resolved metadata key value pair. To learn exactly what steps AzCopy takes to rename object keys, see the <a href="#">How AzCopy renames object keys</a> section below. If AzCopy is unable to rename the key, then the object won't be copied.

### How AzCopy renames object keys

AzCopy performs these steps:

1. Replaces invalid characters with '\_'.
2. Adds the string `rename_` to the beginning of a new valid key.

This key will be used to save the original metadata **value**.

3. Adds the string `rename_key_` to the beginning of a new valid key. This key will be used to save original metadata **invalid key**. You can use this key to try to recover the metadata in Azure side since metadata key is preserved as a value on the Blob storage service.

## Next steps

Find more examples in these articles:

- [Examples: Upload](#)
- [Examples: Download](#)
- [Examples: Copy between accounts](#)
- [Examples: Synchronize](#)
- [Examples: Amazon S3 buckets](#)
- [Examples: Azure Files](#)
- [Tutorial: Migrate on-premises data to cloud storage by using AzCopy](#)

See these articles to configure settings, optimize performance, and troubleshoot issues:

- [AzCopy configuration settings](#)
- [Optimize the performance of AzCopy](#)
- [Find errors and resume jobs by using log and plan files in AzCopy](#)
- [Troubleshoot problems with AzCopy v10](#)

# Troubleshoot problems with AzCopy v10

8/22/2022 • 8 minutes to read • [Edit Online](#)

This article describes common issues that you might encounter while using AzCopy, helps you to identify the causes of those issues, and then suggests ways to resolve them.

## Identifying problems

You can determine whether a job succeeds by looking at the exit code.

If the exit code is `0-success`, then the job completed successfully.

If the exit code is `1-error`, then examine the log file. Once you understand the exact error message, then it becomes much easier to search for the right key words and figure out the solution. To learn more, see [Find errors and resume jobs by using log and plan files in AzCopy](#).

If the exit code is `2-panic`, then check the log file exists. If the file doesn't exist, file a bug or reach out to support.

If the exit code is any other non-zero exit code, it may be an exit code from the system. For example, OOMKilled. Check your operating system documentation for special exit codes.

## 403 errors

It's common to encounter 403 errors. Sometimes they're benign and don't result in failed transfer. For example, in AzCopy logs, you might see that a HEAD request received 403 errors. Those errors appear when AzCopy checks whether a resource is public. In most cases, you can ignore those instances.

In some cases 403 errors can result in a failed transfer. If this happens, other attempts to transfer files will likely fail until you resolve the issue. 403 errors can occur as a result of authentication and authorization issues. They can also occur when requests are blocked due to the storage account firewall configuration.

### Authentication / Authorization issues

403 errors that prevent data transfer occur because of issues with SAS tokens, role based access control (Azure RBAC) roles, and access control list (ACL) configurations.

#### SAS tokens

If you're using a shared access signature (SAS) token, verify the following:

- The expiration and start times of the SAS token are appropriate.
- You selected all the necessary permissions for the token.
- You generated the token by using an official SDK or tool. Try Storage Explorer if you haven't already.

#### Azure RBAC

If you're using role based access control (Azure RBAC) roles via the `azcopy login` command, verify that you have the appropriate Azure roles assigned to your identity (For example: the Storage Blob Data Contributor role).

To learn more about Azure roles, see [Assign an Azure role for access to blob data](#).

#### ACLs

If you're using access control lists (ACLs), verify that your identity appears in an ACL entry for each file or directory that you intend to access. Also, make sure that each ACL entry reflects the appropriate permission level.

To learn more about ACLs and ACL entries, see [Access control lists \(ACLs\) in Azure Data Lake Storage Gen2](#).

To learn about how to incorporate Azure roles together with ACLs, and how system evaluates them to make authorization decisions, see [Access control model in Azure Data Lake Storage Gen2](#).

## Firewall and private endpoint issues

If the storage firewall configuration isn't configured to allow access from the machine where AzCopy is running, AzCopy operations will return an HTTP 403 error.

### Transferring data from or to a local machine

If you're uploading or downloading data between a storage account and an on-premises machine, make sure that the machine that runs AzCopy is able to access either the source or destination storage account. You might have to use IP network rules in the firewall settings of either the source **or** destination accounts to allow access from the public IP address of the machine.

### Transferring data between storage accounts

403 authorization errors can prevent you from transferring data between accounts by using the client machine where AzCopy is running.

If you're copying data between storage accounts, make sure that the machine that runs AzCopy is able to access both the source **and** the destination account. You might have to use IP network rules in the firewall settings of both the source and destination accounts to allow access from the public IP address of the machine. The service will use the IP address of the AzCopy client machine to authorize the source to destination traffic. To learn how to add a public IP address to the firewall settings of a storage account, see [Grant access from an internet IP range](#).

In case your VM doesn't or can't have a public IP address, consider using a private endpoint. See [Use private endpoints for Azure Storage](#).

### Using a Private link

A [Private Link](#) is at the virtual network (VNet) / subnet level. If you want AzCopy requests to go through a Private Link, then AzCopy must make those requests from a VM running in that VNet / subnet. For example, if you configure a Private Link in VNet1 / Subnet1 but the VM on which AzCopy runs is in VNet1 / Subnet2, then AzCopy requests won't use the Private Link and they're expected to fail.

## Proxy-related errors

If you encounter TCP errors such as `dial tcp: lookup proxy.x.x: no such host`, it means that your environment isn't configured to use the correct proxy, or you're using an advanced proxy that AzCopy doesn't recognize.

You need to update the proxy settings to reflect the correct configurations. See [Configure proxy settings](#).

You can also bypass the proxy by setting the environment variable `NO_PROXY="*"`.

Here are the endpoints that AzCopy needs to use:

LOG IN ENDPOINTS	AZURE STORAGE ENDPOINTS
<code>login.microsoftonline.com</code> (global Azure)	<code>(blob \  file \  dfs).core.windows.net</code> (global Azure)
<code>login.chinacloudapi.cn</code> (Azure China)	<code>(blob \  file \  dfs).core.chinacloudapi.cn</code> (Azure China)
<code>login.microsoftonline.de</code> (Azure Germany)	<code>(blob \  file \  dfs).core.cloudapi.de</code> (Azure Germany)

LOG IN ENDPOINTS	AZURE STORAGE ENDPOINTS
<code>login.microsoftonline.us (Azure US Government)</code>	<code>(blob \  file \  dfs).core.usgovcloudapi.net (Azure US Government)</code>

## x509: certificate signed by unknown authority

This error is often related to the use of a proxy, which is using a Secure Sockets Layer (SSL) certificate that isn't trusted by the operating system. Verify your settings and make sure that the certificate is trusted at the operating system level.

We recommend adding the certificate to your machine's root certificate store as that's where the trusted authorities are kept.

## Unrecognized Parameters

If you receive an error message stating that your parameters aren't recognized, make sure that you're using the correct version of AzCopy. AzCopy V8 and earlier versions are deprecated. [AzCopy V10](#) is the current version, and it's a complete rewrite that doesn't share any syntax with the previous versions. Refer to this migration guide [here](#).

Also, make sure to utilize built-in help messages by using the `-h` switch with any command (For example: `azcopy copy -h`). See [Get command help](#). To view the same information online, see [azcopy copy](#).

To help you understand commands, we provide an education tool located [here](#). This tool demonstrates the most popular AzCopy commands along with the most popular command flags. Our examples are [here](#). If you have any question, try searching through existing [GitHub issues](#) first to see if it was answered already.

## Conditional access policy error

You can receive the following error when you invoke the `azcopy login` command.

```
"Failed to perform login command: failed to login with tenantID "common", Azure directory endpoint
"https://login.microsoftonline.com", autorest/adal/devicetoken: -REDACTED- AADSTS50005: User tried to log in
to a device from a platform (Unknown) that's currently not supported through Conditional Access policy.
Supported device platforms are: iOS, Android, Mac, and Windows flavors. Trace ID: -REDACTED- Correlation ID: -
REDACTED- Timestamp: 2021-01-05 01:58:28Z"
```

This error means that your administrator has configured a conditional access policy that specifies what type of device you can log in from. AzCopy uses the device code flow, which can't guarantee that the machine where you're using the AzCopy tool is also where you're logging in.

If your device is among the list of supported platforms, then you might be able to use Storage Explorer, which integrates AzCopy for all data transfers (it passes tokens to AzCopy via the secret store) but provides a login workflow that supports passing device information. AzCopy itself also supports managed identities and service principals, which could be used as an alternative.

If your device isn't among the list of supported platforms, contact your administrator for help.

## Server busy, network errors, timeouts

If you see a large number of failed requests with the `503 Server Busy` status, then your requests are being throttled by the storage service. If you're seeing network errors or timeouts, you might be attempting to push through too much data across your infrastructure and that infrastructure is having difficulty handling it. In all cases, the workaround is similar.

If you see a large file failing over and over again due to certain chunks failing each time, then try to limit the concurrent network connections or throughput limit depending on your specific case. We suggest that you first lower the performance drastically at first, observe whether it solved the initial problem, then ramp up performance again until an overall balance is achieved.

For more information, see [Optimize the performance of AzCopy with Azure Storage](#)

If you're copying data between accounts by using AzCopy, the quality and reliability of the network from where you run AzCopy might impact the overall performance. Even though data transfers from server to server, AzCopy does initiate calls for each file to copy between service endpoints.

## Known constraints with AzCopy

- Copying data from government clouds to commercial clouds isn't supported. However, copying data from commercial clouds to government clouds is supported.
- Asynchronous service-side copy isn't supported. AzCopy performs synchronous copy only. In other words, by the time the job finishes, the data has been moved.
- When copying to an Azure File share, if you forgot to specify the flag `--preserve-smb-permissions`, and you do not want to transfer the data again, then consider using Robocopy to bring over the permissions.
- Azure Functions has a different endpoint for MSI authentication, which AzCopy doesn't yet support.

## Known temporary issues

There's a service issue impacting AzCopy 10.11+ which are using the [PutBlobFromURL API](#) to copy blobs smaller than the given block size (whose default is 8 MiB). If the user has any firewall (VNet / IP / PL / SE Policy) on the source account, the `PutBlobFromURL` API might mistakenly return the message

`409 Copy source blob has been modified`. The workaround is to use AzCopy 10.10.

- [https://azcopyvnext.azureedge.net/release20210415/azcopy\\_darwin\\_amd64\\_10.10.0.zip](https://azcopyvnext.azureedge.net/release20210415/azcopy_darwin_amd64_10.10.0.zip)
- [https://azcopyvnext.azureedge.net/release20210415/azcopy\\_linux\\_amd64\\_10.10.0.tar.gz](https://azcopyvnext.azureedge.net/release20210415/azcopy_linux_amd64_10.10.0.tar.gz)
- [https://azcopyvnext.azureedge.net/release20210415/azcopy\\_windows\\_386\\_10.10.0.zip](https://azcopyvnext.azureedge.net/release20210415/azcopy_windows_386_10.10.0.zip)
- [https://azcopyvnext.azureedge.net/release20210415/azcopy\\_windows\\_amd64\\_10.10.0.zip](https://azcopyvnext.azureedge.net/release20210415/azcopy_windows_amd64_10.10.0.zip)

## See also

- [Get started with AzCopy](#)
- [Find errors and resume jobs by using log and plan files in AzCopy](#)

# What is BlobFuse2? (preview)

8/22/2022 • 5 minutes to read • [Edit Online](#)

## IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

If you need to use BlobFuse in a production environment, BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 setup documentation](#)
- [The BlobFuse v1 project on GitHub](#)

## Overview

BlobFuse2 is a virtual file system driver for Azure Blob storage. It allows you to access your existing Azure block blob data in your storage account through the Linux file system. BlobFuse2 also supports storage accounts with a hierarchical namespace enabled.

### About the BlobFuse2 open source project

BlobFuse2 is an open source project that uses the libfuse open source library (fuse3) to communicate with the Linux FUSE kernel module, and implements the filesystem operations using the Azure Storage REST APIs.

The open source BlobFuse2 project can be found on GitHub:

- [BlobFuse2 home page](#)
- [Blobfuse2 README](#)
- [Report BlobFuse2 issues](#)

### Licensing

This project is [licensed under MIT](#).

## Features

A full list of BlobFuse2 features is in the [BlobFuse2 README](#). Some key features are:

- Mount an Azure storage blob container or Data Lake Storage Gen2 file system on Linux
- Use basic file system operations, such as mkdir, opendir, readdir, rmdir, open, read, create, write, close, unlink, truncate, stat, and rename
- Local caching to improve subsequent access times
- Streaming to support reading and writing large files
- Parallel downloads and uploads to improve access time for large files
- Multiple mounts to the same container for read-only workloads

### BlobFuse2 Enhancements

Blobfuse2 has more feature support and improved performance in multiple user scenarios over v1. For the extensive list of improvements, see [the BlobFuse2 README](#). A summary of enhancements is provided below:

- Improved caching

- More management support through new Azure CLI commands
- Additional logging support
- Compatibility and upgrade options for existing BlobFuse v1 users
- Version checking and upgrade prompting
- Support for configuration file encryption

For a list of performance enhancements over v1, see [the BlobFuse2 README](#) for more details.

### For existing BlobFuse v1 users

The enhancements provided by BlobFuse2 are compelling reasons for upgrading and migrating to BlobFuse2. However, if you aren't ready to migrate, you can [use BlobFuse2 to mount a blob container by using the same configuration options and Azure CLI parameters you used with BlobFuse v1](#).

The [Blobfuse2 Migration Guide](#) provides all of the details you need for compatibility and migrating your existing workloads.

## Support

BlobFuse2 is supported by Microsoft provided that it's used within the specified [limits](#). If you encounter an issue in this preview version, [report it on GitHub](#).

## Limitations

BlobFuse2 doesn't guarantee 100% POSIX compliance as it simply translates requests into [Blob REST APIs](#). For example, rename operations are atomic in POSIX, but not in BlobFuse2.

See [the full list of differences between a native file system and BlobFuse2](#).

### Differences between the Linux file system and BlobFuse2

In many ways, BlobFuse2-mounted storage can be used just like the native Linux file system. The virtual directory scheme is the same with the forward-slash '/' as a delimiter. Basic file system operations, such as mkdir, opendir, readdir, rmdir, open, read, create, write, close, unlink, truncate, stat, and rename work normally.

However, there are some key differences in the way BlobFuse2 behaves:

- **Readdir count of hardlinks:**

For performance reasons, BlobFuse2 does not correctly report the hard links inside a directory. The number of hard links for empty directories is returned as 2. The number for non-empty directories is always returned as 3, regardless of the actual number of hard links.

- **Non-atomic renames:**

Atomic rename operations aren't supported by the Azure Storage Blob Service. Single file renames are actually two operations - a copy, followed by a delete of the original. Directory renames recursively enumerate all files in the directory, and renames each.

- **Special files:**

Blobfuse supports only directories, regular files, and symbolic links. Special files, such as device files, pipes, and sockets aren't supported.

- **mkfifo:**

Fifo creation isn't supported by BlobFuse2. Attempting this action results in a "function not implemented" error.

- **chown and chmod:**

Data Lake Storage Gen2 storage accounts support per object permissions and ACLs, while flat namespace (FNS) block blobs don't. As a result, BlobFuse2 doesn't support the `chown` and `chmod` operations for mounted block blob containers. The operations are supported for Data Lake Storage Gen2.

- **Device files or pipes:**

Creation of device files or pipes isn't supported by BlobFuse2.

- **Extended-attributes (x-attrs):**

BlobFuse2 doesn't support extended-attributes (x-attrs) operations.

## Data integrity

When a file is written to, the data is first persisted into cache on a local disk. The data is written to blob storage only after the file handle is closed. If there's an issue attempting to persist the data to blob storage, you receive an error message.

BlobFuse2 supports both read and write operations. Continuous synchronization of data written to storage by using other APIs or other mounts of BlobFuse2 aren't guaranteed. For data integrity, it's recommended that multiple sources don't modify the same blob, especially at the same time. If one or more applications attempt to write to the same file simultaneously, the results can be unexpected. Depending on the timing of multiple write operations and the freshness of the cache for each, the result could be that the last writer wins and previous writes are lost, or generally that the updated file isn't in the desired state.

### WARNING

In cases where multiple file handles are open to the same file, simultaneous write operations could result in data loss.

## Permissions

When a container is mounted with the default options, all files will get 770 permissions, and only be accessible by the user doing the mounting. If you desire to allow anyone on your machine to access the BlobFuse2 mount, mount it with option "`--allow-other`". This option can also be configured through the `yaml` config file.

As stated previously, the `chown` and `chmod` operations are supported for Data Lake Storage Gen2, but not for flat namespace (FNS) block blobs. Running a 'chmod' operation against a mounted FNS block blob container returns a success message, but the operation won't actually succeed.

## Feature support

This table shows how this feature is supported in your account and the impact on support when you enable certain capabilities.

STORAGE ACCOUNT TYPE	BLOB STORAGE (DEFAULT SUPPORT)	DATA LAKE STORAGE GEN2 <sup>1</sup>	NFS 3.0 <sup>1</sup>	SFTP <sup>1</sup>
Standard general-purpose v2	✓	✓	✓	✓
Premium block blobs	✓	✓	✓	✓

<sup>1</sup> Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, and SSH File Transfer Protocol (SFTP) support all require a storage account with a hierarchical namespace enabled.

## Next steps

- [How to mount an Azure blob storage container on Linux with BlobFuse2 \(preview\)](#)

- [The BlobFuse2 Migration Guide \(from v1\)](#)

## See also

- [BlobFuse2 configuration reference \(preview\)](#)
- [BlobFuse2 command reference \(preview\)](#)
- [How to troubleshoot BlobFuse2 issues \(preview\)](#)

# How to mount an Azure blob storage container on Linux with BlobFuse2 (preview)

8/22/2022 • 5 minutes to read • [Edit Online](#)

BlobFuse2 is a virtual file system driver for Azure Blob storage. BlobFuse2 allows you to access your existing Azure block blob data in your storage account through the Linux file system. For more details see [What is BlobFuse2? \(preview\)](#).

## IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

If you need to use BlobFuse in a production environment, BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 setup documentation](#)
- [The BlobFuse v1 project on GitHub](#)

This guide shows you how to install and configure BlobFuse2, mount an Azure blob container, and access data in the container. The basic steps are:

- [Install BlobFuse2](#)
- [Configure BlobFuse2](#)
- [Mount blob container](#)
- [Access data](#)

## Install BlobFuse2

There are 2 basic options for installing BlobFuse2:

1. [Install BlobFuse2 Binary](#)
2. [Build it from source](#)

### Option 1: Install BlobFuse2 Binary (preferred)

For supported distributions see [the BlobFuse2 releases page](#). For libfuse support information, refer to [the BlobFuse2 README](#).

To check your version of Linux, run the following command:

```
lsb_release -a
```

If there are no binaries available for your distribution, you can [build the binaries from source code](#).

### Install the BlobFuse2 binaries

To install BlobFuse2:

1. Retrieve the latest Blobfuse2 binary for your distro from GitHub, for example:

```
wget https://github.com/Azure/azure-storage-fuse/releases/download/blobfuse2-2.0.0-preview2/blobfuse2-2.0.0-preview.2-ubuntu-20.04-x86-64.deb
```

2. Install BlobFuse2. For example, on an Ubuntu distribution run:

```
sudo apt-get install libfuse3-dev fuse3
sudo apt install blobfuse2-2.0.0-preview.2-ubuntu-20.04-x86-64.deb
```

## Option 2: Build from source

To build the BlobFuse2 binaries from source:

1. Install the dependencies

- a. Install Git:

```
sudo apt-get install git
```

- b. Install BlobFuse2 dependencies: Ubuntu:

```
sudo apt-get install libfuse3-dev fuse3 -y
```

2. Clone the repo

```
git clone https://github.com/Azure/azure-storage-fuse/
cd ./azure-storage-fuse
git checkout main
```

3. Build

```
go get
go build -tags=fuse3
```

### TIP

If you need to install Go, refer to [The download and install page for Go](#).

## Configure BlobFuse2

You can configure BlobFuse2 with a variety of settings. Some of the common settings used include:

- Logging location and options
- Temporary cache file path
- Information about the Azure storage account and blob container to be mounted

The settings can be configured in a yaml configuration file, using environment variables, or as parameters passed to the BlobFuse2 commands. The preferred method is to use the yaml configuration file.

For details about all of the configuration parameters for BlobFuse2, consult the complete reference material for each:

- [Complete BlobFuse2 configuration reference \(preview\)](#)

- [Configuration file reference \(preview\)](#)
- [Environment variable reference \(preview\)](#)
- [Mount command reference \(preview\)](#)

The basic steps for configuring BlobFuse2 in preparation for mounting are:

1. [Configure a temporary path for caching or streaming](#)
2. [Create an empty directory for mounting the blob container](#)
3. [Authorize access to your storage account](#)

### Configure a temporary path for caching

BlobFuse2 provides native-like performance by requiring a temporary path in the file system to buffer and cache any open files. For this temporary path, choose the most performant disk available, or use a ramdisk for the best performance.

#### NOTE

BlobFuse2 stores all open file contents in the temporary path. Make sure to have enough space to accommodate all open files.

#### Choose a caching disk option

There are 3 common options for configuring the temporary path for caching:

- [Use a local high-performing disk](#)
- [Use a ramdisk](#)
- [Use an SSD](#)

#### Use a local high-performing disk

If an existing local disk is chosen for file caching, choose one that will provide the best performance possible, such as an SSD.

#### Use a ramdisk

The following example creates a ramdisk of 16 GB and a directory for BlobFuse2. Choose the size based on your needs. This ramdisk allows BlobFuse2 to open files up to 16 GB in size.

```
sudo mkdir /mnt/ramdisk
sudo mount -t tmpfs -o size=16g tmpfs /mnt/ramdisk
sudo mkdir /mnt/ramdisk/blobfuse2tmp
sudo chown <youruser> /mnt/ramdisk/blobfuse2tmp
```

#### Use an SSD

In Azure, you may use the ephemeral disks (SSD) available on your VMs to provide a low-latency buffer for BlobFuse2. Depending on the provisioning agent used, the ephemeral disk would be mounted on '/mnt' for cloud-init or '/mnt/resource' for waagent VMs.

Make sure your user has access to the temporary path:

```
sudo mkdir /mnt/resource/blobfuse2tmp -p
sudo chown <youruser> /mnt/resource/blobfuse2tmp
```

### Create an empty directory for mounting the blob container

```
mkdir ~/mycontainer
```

### Authorize access to your storage account

You must grant the user mounting the container access to the storage account. The most common ways of doing this are by using:

- A storage account access key
- A shared access signature (SAS)
- A managed identity
- A Service Principal

Authorization information can be provided in a configuration file or in environment variables. For details, see [How to configure Blobfuse2 \(preview\)](#).

## Mount blob container

### IMPORTANT

BlobFuse2 does not support overlapping mount paths. While running multiple instances of BlobFuse2 make sure each instance has a unique and non-overlapping mount point.

BlobFuse2 does not support co-existence with NFS on the same mount path. The behavior resulting from doing so is undefined and could result in data corruption.

To mount an Azure block blob container using BlobFuse2, run the following command. This command mounts the container specified in `./config.yaml` onto the location `~/mycontainer`:

```
blobfuse2 mount ~/mycontainer --config-file=./config.yaml
```

### NOTE

For a full list of mount options, check [the BlobFuse2 mount command reference \(preview\)](#).

You should now have access to your block blobs through the Linux file system and related APIs. To test your deployment try creating a new directory and file:

```
cd ~/mycontainer
mkdir test
echo "hello world" > test/blob.txt
```

## Access data

Generally, you can work with the BlobFuse2-mounted storage like you would work with the native Linux file system. It uses the virtual directory scheme with the forward-slash '/' as a delimiter in the file path and supports basic file system operations such as mkdir, opendir, readdir, rmdir, open, read, create, write, close, unlink, truncate, stat, and rename.

However, there are some [differences in functionality](#) you need to be aware of. Some of the key differences are related to:

- [Differences between the Linux file system and BlobFuse2](#)
- [Data Integrity](#)
- [Permissions](#)

## Feature support

This table shows how this feature is supported in your account and the impact on support when you enable certain capabilities.

STORAGE ACCOUNT TYPE	BLOB STORAGE (DEFAULT SUPPORT)	DATA LAKE STORAGE GEN2 <sup>1</sup>	NFS 3.0 <sup>1</sup>	SFTP <sup>1</sup>
Standard general-purpose v2	✓	✓	✓	✓
Premium block blobs	✓	✓	✓	✓

<sup>1</sup> Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, and SSH File Transfer Protocol (SFTP) support all require a storage account with a hierarchical namespace enabled.

## See also

- [Blobfuse2 Migration Guide \(from BlobFuse v1\)](#)
- [BlobFuse2 configuration reference \(preview\)](#)
- [BlobFuse2 command reference \(preview\)](#)
- [How to troubleshoot BlobFuse2 issues \(preview\)](#)

# How to mount Blob storage as a file system with BlobFuse

8/22/2022 • 4 minutes to read • [Edit Online](#)

## Overview

### NOTE

This article is about the original version of BlobFuse. It is simply referred to as "BlobFuse" in many cases, but is also referred to as "BlobFuse v1" in this and other articles to distinguish it from the next generation of BlobFuse, BlobFuse2. BlobFuse2 is currently in preview and might not be suitable for production workloads.

To learn about the improvements made in BlobFuse2, see [What is BlobFuse2?](#).

**BlobFuse** is a virtual file system driver for Azure Blob storage. BlobFuse allows you to access your existing block blob data in your storage account through the Linux file system. BlobFuse uses the virtual directory scheme with the forward-slash '/' as a delimiter.

This guide shows you how to use BlobFuse, and mount a Blob storage container on Linux and access data. To learn more about BlobFuse, see the [readme](#) and [wiki](#).

### WARNING

BlobFuse doesn't guarantee 100% POSIX compliance as it simply translates requests into [Blob REST APIs](#). For example, rename operations are atomic in POSIX, but not in BlobFuse. For a full list of differences between a native file system and BlobFuse, visit [the BlobFuse source code repository](#).

## Install BlobFuse on Linux

BlobFuse binaries are available on [the Microsoft software repositories for Linux](#) for Ubuntu, Debian, SUSE, CentOS, Oracle Linux and RHEL distributions. To install BlobFuse on those distributions, configure one of the repositories from the list. You can also build the binaries from source code following the [Azure Storage installation steps](#) if there are no binaries available for your distribution.

BlobFuse is published in the Linux repo for Ubuntu versions: 16.04, 18.04, and 20.04, RHEL versions: 7.5, 7.8, 7.9, 8.0, 8.1, 8.2, CentOS versions: 7.0, 8.0, Debian versions: 9.0, 10.0, SUSE version: 15, OracleLinux 8.1 . Run this command to make sure that you have one of those versions deployed:

```
lsb_release -a
```

### Configure the Microsoft package repository

Configure the [Linux Package Repository for Microsoft Products](#).

As an example, on an Enterprise Linux 8 distribution:

```
sudo rpm -Uvh https://packages.microsoft.com/config/rhel/8/packages-microsoft-prod.rpm
```

Similarly, change the URL to `.../rhel/7/...` to point to an Enterprise Linux 7 distribution.

Another example on an Ubuntu 20.04 distribution:

```
wget https://packages.microsoft.com/config/ubuntu/20.04/packages-microsoft-prod.deb
sudo dpkg -i packages-microsoft-prod.deb
sudo apt-get update
```

Similarly, change the URL to `.../ubuntu/16.04/...` or `.../ubuntu/18.04/...` to reference another Ubuntu version.

## Install BlobFuse

On an Ubuntu/Debian distribution:

```
sudo apt-get install blobfuse
```

On an Enterprise Linux distribution:

```
sudo yum install blobfuse
```

On a SUSE distribution:

```
sudo zypper install blobfuse
```

## Prepare for mounting

BlobFuse provides native-like performance by requiring a temporary path in the file system to buffer and cache any open files. For this temporary path, choose the most performant disk, or use a ramdisk for best performance.

### NOTE

BlobFuse stores all open file contents in the temporary path. Make sure to have enough space to accommodate all open files.

### (Optional) Use a ramdisk for the temporary path

The following example creates a ramdisk of 16 GB and a directory for BlobFuse. Choose the size based on your needs. This ramdisk allows BlobFuse to open files up to 16 GB in size.

```
sudo mkdir /mnt/ramdisk
sudo mount -t tmpfs -o size=16g tmpfs /mnt/ramdisk
sudo mkdir /mnt/ramdisk/blobfusetmp
sudo chown <youruser> /mnt/ramdisk/blobfusetmp
```

### Use an SSD as a temporary path

In Azure, you may use the ephemeral disks (SSD) available on your VMs to provide a low-latency buffer for BlobFuse. Depending on the provisioning agent used, the ephemeral disk would be mounted on '/mnt' for cloud-init or '/mnt/resource' for waagent VMs.

Make sure your user has access to the temporary path:

```
sudo mkdir /mnt/resource/blobfusetmp -p
sudo chown <youruser> /mnt/resource/blobfusetmp
```

## Authorize access to your storage account

You can authorize access to your storage account by using the account access key, a shared access signature, a managed identity, or a service principal. Authorization information can be provided on the command line, in a config file, or in environment variables. For details, see [Valid authentication setups](#) in the BlobFuse readme.

For example, suppose you are authorizing with the account access keys and storing them in a config file. The config file should have the following format:

```
accountName myaccount
accountKey storageaccesskey
containerName mycontainer
```

The `accountName` is the name of your storage account, and not the full URL.

Create this file using:

```
touch /path/to/fuse_connection.cfg
```

Once you've created and edited this file, make sure to restrict access so no other users can read it.

```
chmod 600 /path/to/fuse_connection.cfg
```

### NOTE

If you have created the configuration file on Windows, make sure to run `dos2unix` to sanitize and convert the file to Unix format.

## Create an empty directory for mounting

```
mkdir ~/mycontainer
```

## Mount

### NOTE

For a full list of mount options, check [the BlobFuse repository](#).

To mount BlobFuse, run the following command with your user. This command mounts the container specified in '/path/to/fuse\_connection.cfg' onto the location '/mycontainer'.

```
blobfuse ~/mycontainer --tmp-path=/mnt/resource/blobfusetmp --config-file=/path/to/fuse_connection.cfg -o
attr_timeout=240 -o entry_timeout=240 -o negative_timeout=120
```

#### NOTE

If you use an ADLS account, you must include `--use-adls=true`.

You should now have access to your block blobs through the regular file system APIs. The user who mounts the directory is the only person who can access it, by default, which secures the access. To allow access to all users, you can mount via the option `-o allow_other`.

```
cd ~/mycontainer
mkdir test
echo "hello world" > test/blob.txt
```

## Persist the mount

To learn how to persist the mount, see [Persisting](#) in the BlobFuse wiki.

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## Next steps

- [BlobFuse home page](#)
- [Report BlobFuse issues](#)

# Troubleshooting BlobFuse2 (preview)

8/22/2022 • 2 minutes to read • [Edit Online](#)

This article provides references to assist in troubleshooting BlobFuse2 issues during the public preview.

## The troubleshooting guide

During the preview of BlobFuse2, refer to [The BlobFuse2 Troubleshoot Guide \(TSG\) on GitHub](#)

### IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

If you need to use BlobFuse in a production environment, BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 setup documentation](#)
- [The BlobFuse v1 project on GitHub](#)

## See also

- [What is BlobFuse2? \(preview\)](#)
- [How to mount an Azure blob storage container on Linux with BlobFuse2 \(preview\)](#)
- [BlobFuse2 configuration reference \(preview\)](#)
- [BlobFuse2 command reference \(preview\)](#)

# Copy and transform data in Azure Blob storage by using Azure Data Factory or Azure Synapse Analytics

8/22/2022 • 36 minutes to read • [Edit Online](#)

APPLIES TO: Azure Data Factory Azure Synapse Analytics

This article outlines how to use the Copy activity in Azure Data Factory and Azure Synapse pipelines to copy data from and to Azure Blob storage. It also describes how to use the Data Flow activity to transform data in Azure Blob storage. To learn more read the [Azure Data Factory](#) and the [Azure Synapse Analytics](#) introduction articles.

## TIP

To learn about a migration scenario for a data lake or a data warehouse, see the article [Migrate data from your data lake or data warehouse to Azure](#).

## Supported capabilities

This Azure Blob storage connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR	MANAGED PRIVATE ENDPOINT
<a href="#">Copy activity</a> (source/sink)	① ②	✓ Exclude storage account V1
<a href="#">Mapping data flow</a> (source/sink)	①	✓ Exclude storage account V1
<a href="#">Lookup activity</a>	① ②	✓ Exclude storage account V1
<a href="#">GetMetadata activity</a>	① ②	✓ Exclude storage account V1
<a href="#">Delete activity</a>	① ②	✓ Exclude storage account V1

① Azure integration runtime ② Self-hosted integration runtime

For the Copy activity, this Blob storage connector supports:

- Copying blobs to and from general-purpose Azure storage accounts and hot/cool blob storage.
- Copying blobs by using an account key, a service shared access signature (SAS), a service principal, or managed identities for Azure resource authentications.
- Copying blobs from block, append, or page blobs and copying data to only block blobs.
- Copying blobs as is, or parsing or generating blobs with [supported file formats and compression codecs](#).
- [Preserving file metadata during copy](#).

## Get started

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- [The Copy Data tool](#)
- [The Azure portal](#)
- [The .NET SDK](#)
- [The Python SDK](#)
- [Azure PowerShell](#)
- [The REST API](#)
- [The Azure Resource Manager template](#)

## Create an Azure Blob Storage linked service using UI

Use the following steps to create an Azure Blob Storage linked service in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

- [Azure Data Factory](#)
- [Azure Synapse](#)

The screenshot shows the Microsoft Azure Data Factory interface. The top navigation bar includes 'Microsoft Azure', 'Data Factory', 'YourDataFactory', and user information ('user@contoso.com CONTOSO, LTD.'). The left sidebar has several sections: 'Connections' (with 'Linked services' selected and highlighted by a red box), 'Integration runtimes', 'Azure Purview (Preview)', 'Source control' (with a red box around its icon), 'Author' (with 'Triggers' and 'Global parameters'), 'Security' (with 'Customer managed key' and 'Credentials'), and 'Managed private endpoints'. The main content area is titled 'Linked services' and contains a message: 'Linked service defines the connection information to a data store or compute. Learn more'. Below this is a '+ New' button (also highlighted by a red box) and a 'Filter by name' input field. A table header with columns 'Name ↑↓', 'Type ↑↓', 'Related ↑↓', and 'Annotations ↑↓' is shown, followed by a message 'Showing 0 - 0 of 0 items' and a large 'No linked service to show' icon. At the bottom, there is a 'Create linked service' button.

2. Search for blob and select the Azure Blob Storage connector.

## New linked service

Data store Compute

 blob

All Azure Database File Generic protocol NoSQL Services and apps



Azure Blob Storage

Continue

Cancel

3. Configure the service details, test the connection, and create the new linked service.

## New linked service (Azure Blob Storage)

Name \*

Description

Connect via integration runtime \* ⓘ

▼

Authentication method

▼

Connection string Azure Key Vault

Account selection method ⓘ

From Azure subscription  Enter manually

Azure subscription ⓘ

▼

Storage account name \*

▼ ↻

Additional connection properties

+ New

---

Test connection ⓘ

To linked service  To file path

Annotations

+ New

---

▷ Parameters

▷ Advanced ⓘ

---

Create Back Test connection Cancel

## Connector configuration details

The following sections provide details about properties that are used to define Data Factory and Synapse pipeline entities specific to Blob storage.

## Linked service properties

This Blob storage connector supports the following authentication types. See the corresponding sections for details.

- [Account key authentication](#)
- [Shared access signature authentication](#)
- [Service principal authentication](#)
- [System-assigned managed identity authentication](#)
- [User-assigned managed identity authentication](#)

**NOTE**

- If you want to use the public Azure integration runtime to connect to your Blob storage by leveraging the **Allow trusted Microsoft services to access this storage account** option enabled on Azure Storage firewall, you must use **managed identity authentication**. For more information about the Azure Storage firewalls settings, see [Configure Azure Storage firewalls and virtual networks](#).
- When you use PolyBase or COPY statement to load data into Azure Synapse Analytics, if your source or staging Blob storage is configured with an Azure Virtual Network endpoint, you must use managed identity authentication as required by Azure Synapse. See the [Managed identity authentication](#) section for more configuration prerequisites.

**NOTE**

Azure HDInsight and Azure Machine Learning activities only support authentication that uses Azure Blob storage account keys.

## Account key authentication

The following properties are supported for storage account key authentication in Azure Data Factory or Synapse pipelines:

PROPERTY	DESCRIPTION	REQUIRED
type	The <code>type</code> property must be set to <code>AzureBlobStorage</code> (suggested) or <code>AzureStorage</code> (see the following notes).	Yes
connectionString	Specify the information needed to connect to Storage for the <code>connectionString</code> property. You can also put the account key in Azure Key Vault and pull the <code>accountKey</code> configuration out of the connection string. For more information, see the following samples and the <a href="#">Store credentials in Azure Key Vault</a> article.	Yes
connectVia	The <a href="#">integration runtime</a> to be used to connect to the data store. You can use the Azure integration runtime or the self-hosted integration runtime (if your data store is in a private network). If this property isn't specified, the service uses the default Azure integration runtime.	No

**NOTE**

A secondary Blob service endpoint is not supported when you're using account key authentication. You can use other authentication types.

## NOTE

If you're using the `AzureStorage` type linked service, it's still supported as is. But we suggest that you use the new `AzureBlobStorage` linked service type going forward.

## Example:

```
{
 "name": "AzureBlobStorageLinkedService",
 "properties": {
 "type": "AzureBlobStorage",
 "typeProperties": {
 "connectionString": "DefaultEndpointsProtocol=https;AccountName=<accountname>;AccountKey=<accountkey>"
 },
 "connectVia": {
 "referenceName": "<name of Integration Runtime>",
 "type": "IntegrationRuntimeReference"
 }
 }
}
```

## Example: store the account key in Azure Key Vault

```
{
 "name": "AzureBlobStorageLinkedService",
 "properties": {
 "type": "AzureBlobStorage",
 "typeProperties": {
 "connectionString": "DefaultEndpointsProtocol=https;AccountName=<accountname>;",
 "accountKey": {
 "type": "AzureKeyVaultSecret",
 "store": {
 "referenceName": "<Azure Key Vault linked service name>",
 "type": "LinkedServiceReference"
 },
 "secretName": "<secretName>"
 }
 },
 "connectVia": {
 "referenceName": "<name of Integration Runtime>",
 "type": "IntegrationRuntimeReference"
 }
 }
}
```

## Shared access signature authentication

A shared access signature provides delegated access to resources in your storage account. You can use a shared access signature to grant a client limited permissions to objects in your storage account for a specified time.

You don't have to share your account access keys. The shared access signature is a URI that encompasses in its query parameters all the information necessary for authenticated access to a storage resource. To access storage resources with the shared access signature, the client only needs to pass in the shared access signature to the appropriate constructor or method.

For more information about shared access signatures, see [Shared access signatures: Understand the shared access signature model](#).

**NOTE**

- The service now supports both *service shared access signatures* and *account shared access signatures*. For more information about shared access signatures, see [Grant limited access to Azure Storage resources using shared access signatures](#).
- In later dataset configurations, the folder path is the absolute path starting from the container level. You need to configure one aligned with the path in your SAS URI.

The following properties are supported for using shared access signature authentication:

PROPERTY	DESCRIPTION	REQUIRED
type	The <code>type</code> property must be set to <code>AzureBlobStorage</code> (suggested) or <code>AzureStorage</code> (see the following note).	Yes
sasUri	Specify the shared access signature URI to the Storage resources such as blob or container. Mark this field as <code>SecureString</code> to store it securely. You can also put the SAS token in Azure Key Vault to use auto-rotation and remove the token portion. For more information, see the following samples and <a href="#">Store credentials in Azure Key Vault</a> .	Yes
connectVia	The <a href="#">integration runtime</a> to be used to connect to the data store. You can use the Azure integration runtime or the self-hosted integration runtime (if your data store is in a private network). If this property isn't specified, the service uses the default Azure integration runtime.	No

**NOTE**

If you're using the `AzureStorage` type linked service, it's still supported as is. But we suggest that you use the new `AzureBlobStorage` linked service type going forward.

**Example:**

```
{
 "name": "AzureBlobStorageLinkedService",
 "properties": {
 "type": "AzureBlobStorage",
 "typeProperties": {
 "sasUri": {
 "type": "SecureString",
 "value": "<SAS URI of the Azure Storage resource e.g.

https://<accountname>.blob.core.windows.net/?sv=<storage version>&st=<start time>&se=<expire time>&sr=<resource>&sp=<permissions>&sip=<ip range>&spr=<protocol>&sig=<signature>>"
 }
 },
 "connectVia": {
 "referenceName": "<name of Integration Runtime>",
 "type": "IntegrationRuntimeReference"
 }
 }
}
```

### Example: store the account key in Azure Key Vault

```
{
 "name": "AzureBlobStorageLinkedService",
 "properties": {
 "type": "AzureBlobStorage",
 "typeProperties": {
 "sasUri": {
 "type": "SecureString",
 "value": "<SAS URI of the Azure Storage resource without token e.g.

https://<accountname>.blob.core.windows.net/>"
 },
 "sasToken": {
 "type": "AzureKeyVaultSecret",
 "store": {
 "referenceName": "<Azure Key Vault linked service name>",
 "type": "LinkedServiceReference"
 },
 "secretName": "<secretName with value of SAS token e.g. ?sv=<storage version>&st=<start

time>&se=<expire time>&sr=<resource>&sp=<permissions>&sip=<ip range>&spr=<protocol>&sig=<signature>>"
 }
 },
 "connectVia": {
 "referenceName": "<name of Integration Runtime>",
 "type": "IntegrationRuntimeReference"
 }
 }
}
```

When you create a shared access signature URI, consider the following points:

- Set appropriate read/write permissions on objects based on how the linked service (read, write, read/write) is used.
- Set **Expiry time** appropriately. Make sure that the access to Storage objects doesn't expire within the active period of the pipeline.
- The URI should be created at the right container or blob based on the need. A shared access signature URI to a blob allows the data factory or Synapse pipeline to access that particular blob. A shared access signature URI to a Blob storage container allows the data factory or Synapse pipeline to iterate through blobs in that container. To provide access to more or fewer objects later, or to update the shared access signature URI, remember to update the linked service with the new URI.

### Service principal authentication

For general information about Azure Storage service principal authentication, see [Authenticate access to Azure Storage using Azure Active Directory](#).

To use service principal authentication, follow these steps:

1. Register an application entity in Azure Active Directory (Azure AD) by following [Register your application with an Azure AD tenant](#). Make note of these values, which you use to define the linked service:
  - Application ID
  - Application key
  - Tenant ID
2. Grant the service principal proper permission in Azure Blob storage. For more information on the roles, see [Use the Azure portal to assign an Azure role for access to blob and queue data](#).
  - As source, in Access control (IAM), grant at least the **Storage Blob Data Reader** role.
  - As sink, in Access control (IAM), grant at least the **Storage Blob Data Contributor** role.

These properties are supported for an Azure Blob storage linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The <b>type</b> property must be set to <b>AzureBlobStorage</b> .	Yes
serviceEndpoint	Specify the Azure Blob storage service endpoint with the pattern of <code>https://&lt;accountName&gt;.blob.core.windows.net/</code>	Yes
accountKind	Specify the kind of your storage account. Allowed values are: <b>Storage</b> (general purpose v1), <b>StorageV2</b> (general purpose v2), <b>BlobStorage</b> , or <b>BlockBlobStorage</b> .  When using Azure Blob linked service in data flow, managed identity or service principal authentication is not supported when account kind as empty or "Storage". Specify the proper account kind, choose a different authentication, or upgrade your storage account to general purpose v2.	No
servicePrincipalId	Specify the application's client ID.	Yes
servicePrincipalKey	Specify the application's key. Mark this field as <b>SecureString</b> to store it securelyFactory, or <a href="#">reference a secret stored in Azure Key Vault</a> .	Yes
tenant	Specify the tenant information (domain name or tenant ID) under which your application resides. Retrieve it by hovering over the upper-right corner of the Azure portal.	Yes

PROPERTY	DESCRIPTION	REQUIRED
azureCloudType	For service principal authentication, specify the type of Azure cloud environment, to which your Azure Active Directory application is registered. Allowed values are <b>AzurePublic</b> , <b>AzureChina</b> , <b>AzureUsGovernment</b> , and <b>AzureGermany</b> . By default, the data factory or Synapse pipeline's cloud environment is used.	No
connectVia	The <a href="#">integration runtime</a> to be used to connect to the data store. You can use the Azure integration runtime or the self-hosted integration runtime (if your data store is in a private network). If this property isn't specified, the service uses the default Azure integration runtime.	No

#### NOTE

- If your blob account enables [soft delete](#), service principal authentication is not supported in Data Flow.
- If you access the blob storage through private endpoint using Data Flow, note when service principal authentication is used Data Flow connects to the ADLS Gen2 endpoint instead of Blob endpoint. Make sure you create the corresponding private endpoint in your data factory or Synapse workspace to enable access.

#### NOTE

Service principal authentication is supported only by the "AzureBlobStorage" type linked service, not the previous "AzureStorage" type linked service.

#### Example:

```
{
 "name": "AzureBlobStorageLinkedService",
 "properties": {
 "type": "AzureBlobStorage",
 "typeProperties": {
 "serviceEndpoint": "https://<accountName>.blob.core.windows.net/",
 "accountKind": "StorageV2",
 "servicePrincipalId": "<service principal id>",
 "servicePrincipalKey": {
 "type": "SecureString",
 "value": "<service principal key>"
 },
 "tenant": "<tenant info, e.g. microsoft.onmicrosoft.com>"
 },
 "connectVia": {
 "referenceName": "<name of Integration Runtime>",
 "type": "IntegrationRuntimeReference"
 }
 }
}
```

#### System-assigned managed identity authentication

A data factory or Synapse pipeline can be associated with a [system-assigned managed identity for Azure resources](#), which represents that resource for authentication to other Azure services. You can directly use this system-assigned managed identity for Blob storage authentication, which is similar to using your own service principal. It allows this designated resource to access and copy data from or to Blob storage. To learn more about managed identities for Azure resources, see [Managed identities for Azure resources](#)

For general information about Azure Storage authentication, see [Authenticate access to Azure Storage using Azure Active Directory](#). To use managed identities for Azure resource authentication, follow these steps:

1. [Retrieve system-assigned managed identity information](#) by copying the value of the system-assigned managed identity object ID generated along with your factory or Synapse workspace.
2. Grant the managed identity permission in Azure Blob storage. For more information on the roles, see [Use the Azure portal to assign an Azure role for access to blob and queue data](#).
  - **As source**, in [Access control \(IAM\)](#), grant at least the **Storage Blob Data Reader** role.
  - **As sink**, in [Access control \(IAM\)](#), grant at least the **Storage Blob Data Contributor** role.

These properties are supported for an Azure Blob storage linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The <b>type</b> property must be set to <b>AzureBlobStorage</b> .	Yes
serviceEndpoint	Specify the Azure Blob storage service endpoint with the pattern of <code>https://&lt;accountName&gt;.blob.core.windows.net/</code>	Yes
accountKind	Specify the kind of your storage account. Allowed values are: <b>Storage</b> (general purpose v1), <b>StorageV2</b> (general purpose v2), <b>BlobStorage</b> , or <b>BlockBlobStorage</b> .  When using Azure Blob linked service in data flow, managed identity or service principal authentication is not supported when account kind as empty or "Storage". Specify the proper account kind, choose a different authentication, or upgrade your storage account to general purpose v2.	No
connectVia	The <a href="#">integration runtime</a> to be used to connect to the data store. You can use the Azure integration runtime or the self-hosted integration runtime (if your data store is in a private network). If this property isn't specified, the service uses the default Azure integration runtime.	No

**Example:**

```
{
 "name": "AzureBlobStorageLinkedService",
 "properties": {
 "type": "AzureBlobStorage",
 "typeProperties": {
 "serviceEndpoint": "https://<accountName>.blob.core.windows.net/",
 "accountKind": "StorageV2"
 },
 "connectVia": {
 "referenceName": "<name of Integration Runtime>",
 "type": "IntegrationRuntimeReference"
 }
 }
}
```

## User-assigned managed identity authentication

A data factory can be assigned with one or multiple [user-assigned managed identities](#). You can use this user-assigned managed identity for Blob storage authentication, which allows to access and copy data from or to Blob storage. To learn more about managed identities for Azure resources, see [Managed identities for Azure resources](#)

For general information about Azure storage authentication, see [Authenticate access to Azure Storage using Azure Active Directory](#). To use user-assigned managed identity authentication, follow these steps:

1. Create one or multiple user-assigned managed identities and grant permission in Azure Blob storage. For more information on the roles, see [Use the Azure portal to assign an Azure role for access to blob and queue data](#).
  - As source, in **Access control (IAM)**, grant at least the **Storage Blob Data Reader** role.
  - As sink, in **Access control (IAM)**, grant at least the **Storage Blob Data Contributor** role.
2. Assign one or multiple user-assigned managed identities to your data factory and [create credentials](#) for each user-assigned managed identity.

These properties are supported for an Azure Blob storage linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The <b>type</b> property must be set to <b>AzureBlobStorage</b> .	Yes
serviceEndpoint	Specify the Azure Blob storage service endpoint with the pattern of <code>https://&lt;accountName&gt;.blob.core.windows.net/</code>	Yes

PROPERTY	DESCRIPTION	REQUIRED
accountKind	<p>Specify the kind of your storage account. Allowed values are: <b>Storage</b> (general purpose v1), <b>StorageV2</b> (general purpose v2), <b>BlobStorage</b>, or <b>BlockBlobStorage</b>.</p> <p>When using Azure Blob linked service in data flow, managed identity or service principal authentication is not supported when account kind as empty or "Storage". Specify the proper account kind, choose a different authentication, or upgrade your storage account to general purpose v2.</p>	No
credentials	Specify the user-assigned managed identity as the credential object.	Yes
connectVia	The <a href="#">integration runtime</a> to be used to connect to the data store. You can use the Azure integration runtime or the self-hosted integration runtime (if your data store is in a private network). If this property isn't specified, the service uses the default Azure integration runtime.	No

### Example:

```
{
 "name": "AzureBlobStorageLinkedService",
 "properties": {
 "type": "AzureBlobStorage",
 "typeProperties": {
 "serviceEndpoint": "https://<accountName>.blob.core.windows.net/",
 "accountKind": "StorageV2",
 "credential": {
 "referenceName": "credential1",
 "type": "CredentialReference"
 }
 },
 "connectVia": {
 "referenceName": "<name of Integration Runtime>",
 "type": "IntegrationRuntimeReference"
 }
 }
}
```

### IMPORTANT

If you use PolyBase or COPY statement to load data from Blob storage (as a source or as staging) into Azure Synapse Analytics, when you use managed identity authentication for Blob storage, make sure you also follow steps 1 to 3 in [this guidance](#). Those steps will register your server with Azure AD and assign the Storage Blob Data Contributor role to your server. Data Factory handles the rest. If you configure Blob storage with an Azure Virtual Network endpoint, you also need to have **Allow trusted Microsoft services to access this storage account** turned on under Azure Storage account **Firewalls and Virtual networks** settings menu as required by Azure Synapse.

**NOTE**

- If your blob account enables [soft delete](#), system-assigned/user-assigned managed identity authentication is not supported in Data Flow.
- If you access the blob storage through private endpoint using Data Flow, note when system-assigned/user-assigned managed identity authentication is used Data Flow connects to the ADLS Gen2 endpoint instead of Blob endpoint. Make sure you create the corresponding private endpoint in ADF to enable access.

**NOTE**

System-assigned/user-assigned managed identity authentication is supported only by the "AzureBlobStorage" type linked service, not the previous "AzureStorage" type linked service.

## Dataset properties

For a full list of sections and properties available for defining datasets, see the [Datasets](#) article.

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- [Avro format](#)
- [Binary format](#)
- [Delimited text format](#)
- [Excel format](#)
- [JSON format](#)
- [ORC format](#)
- [Parquet format](#)
- [XML format](#)

The following properties are supported for Azure Blob storage under `location` settings in a format-based dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The <b>type</b> property of the location in the dataset must be set to <b>AzureBlobStorageLocation</b> .	Yes
container	The blob container.	Yes
folderPath	The path to the folder under the given container. If you want to use a wildcard to filter the folder, skip this setting and specify that in activity source settings.	No
fileName	The file name under the given container and folder path. If you want to use wildcard to filter files, skip this setting and specify that in activity source settings.	No

**Example:**

```
{
 "name": "DelimitedTextDataset",
 "properties": {
 "type": "DelimitedText",
 "linkedServiceName": {
 "referenceName": "<Azure Blob Storage linked service name>",
 "type": "LinkedServiceReference"
 },
 "schema": [< physical schema, optional, auto retrieved during authoring >],
 "typeProperties": {
 "location": {
 "type": "AzureBlobStorageLocation",
 "container": "containername",
 "folderPath": "folder/subfolder"
 },
 "columnDelimiter": ",",
 "quoteChar": "\"",
 "firstRowAsHeader": true,
 "compressionCodec": "gzip"
 }
 }
}
```

## Copy activity properties

For a full list of sections and properties available for defining activities, see the [Pipelines](#) article. This section provides a list of properties that the Blob storage source and sink support.

### Blob storage as a source type

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- [Avro format](#)
- [Binary format](#)
- [Delimited text format](#)
- [Excel format](#)
- [JSON format](#)
- [ORC format](#)
- [Parquet format](#)
- [XML format](#)

The following properties are supported for Azure Blob storage under `storeSettings` settings in a format-based copy source:

PROPERTY	DESCRIPTION	REQUIRED
type	The <code>type</code> property under <code>storeSettings</code> must be set to <code>AzureBlobStorageReadSettings</code> .	Yes
<i>Locate the files to copy:</i>		
OPTION 1: static path	Copy from the given container or folder/file path specified in the dataset. If you want to copy all blobs from a container or folder, additionally specify <code>wildcardFileName</code> as <code>*</code> .	

PROPERTY	DESCRIPTION	REQUIRED
OPTION 2: blob prefix - prefix	<p>Prefix for the blob name under the given container configured in a dataset to filter source blobs. Blobs whose names start with <code>container_in_dataset/this_prefix</code> are selected. It utilizes the service-side filter for Blob storage, which provides better performance than a wildcard filter.</p> <p>When you use prefix and choose to copy to file-based sink with preserving hierarchy, note the sub-path after the last "/" in prefix will be preserved. For example, you have source <code>container/folder/subfolder/file.txt</code>, and configure prefix as <code>folder/sub</code>, then the preserved file path is <code>subfolder/file.txt</code>.</p>	No
OPTION 3: wildcard - wildcardFolderPath	<p>The folder path with wildcard characters under the given container configured in a dataset to filter source folders.</p> <p>Allowed wildcards are: <code>*</code> (matches zero or more characters) and <code>?</code> (matches zero or single character). Use <code>^</code> to escape if your folder name has wildcard or this escape character inside.</p> <p>See more examples in <a href="#">Folder and file filter examples</a>.</p>	No
OPTION 3: wildcard - wildcardFileName	<p>The file name with wildcard characters under the given container and folder path (or wildcard folder path) to filter source files.</p> <p>Allowed wildcards are: <code>*</code> (matches zero or more characters) and <code>?</code> (matches zero or single character). Use <code>^</code> to escape if your file name has a wildcard or this escape character inside. See more examples in <a href="#">Folder and file filter examples</a>.</p>	Yes
OPTION 4: a list of files - fileListPath	<p>Indicates to copy a given file set. Point to a text file that includes a list of files you want to copy, one file per line, which is the relative path to the path configured in the dataset.</p> <p>When you're using this option, do not specify a file name in the dataset. See more examples in <a href="#">File list examples</a>.</p>	No
<i>Additional settings:</i>		

PROPERTY	DESCRIPTION	REQUIRED
recursive	<p>Indicates whether the data is read recursively from the subfolders or only from the specified folder. Note that when <b>recursive</b> is set to <b>true</b> and the sink is a file-based store, an empty folder or subfolder isn't copied or created at the sink.</p> <p>Allowed values are <b>true</b> (default) and <b>false</b>.</p> <p>This property doesn't apply when you configure <code>fileListPath</code>.</p>	No
deleteFilesAfterCompletion	<p>Indicates whether the binary files will be deleted from source store after successfully moving to the destination store. The file deletion is per file, so when copy activity fails, you will see some files have already been copied to the destination and deleted from source, while others are still remaining on source store.</p> <p>This property is only valid in binary files copy scenario. The default value: false.</p>	No
modifiedDatetimeStart	<p>Files are filtered based on the attribute: last modified.</p> <p>The files will be selected if their last modified time is greater than or equal to <code>modifiedDatetimeStart</code> and less than <code>modifiedDatetimeEnd</code>. The time is applied to a UTC time zone in the format of "2018-12-01T05:00:00Z".</p> <p>The properties can be <b>NULL</b>, which means no file attribute filter will be applied to the dataset. When <code>modifiedDatetimeStart</code> has a datetime value but <code>modifiedDatetimeEnd</code> is <b>NULL</b>, the files whose last modified attribute is greater than or equal to the datetime value will be selected. When <code>modifiedDatetimeEnd</code> has a datetime value but <code>modifiedDatetimeStart</code> is <b>NULL</b>, the files whose last modified attribute is less than the datetime value will be selected.</p> <p>This property doesn't apply when you configure <code>fileListPath</code>.</p>	No
modifiedDatetimeEnd	Same as above.	No
enablePartitionDiscovery	<p>For files that are partitioned, specify whether to parse the partitions from the file path and add them as additional source columns.</p> <p>Allowed values are <b>false</b> (default) and <b>true</b>.</p>	No

PROPERTY	DESCRIPTION	REQUIRED
partitionRootPath	<p>When partition discovery is enabled, specify the absolute root path in order to read partitioned folders as data columns.</p> <p>If it is not specified, by default,</p> <ul style="list-style-type: none"> <li>- When you use file path in dataset or list of files on source, partition root path is the path configured in dataset.</li> <li>- When you use wildcard folder filter, partition root path is the sub-path before the first wildcard.</li> <li>- When you use prefix, partition root path is sub-path before the last "/".</li> </ul> <p>For example, assuming you configure the path in dataset as "root/folder/year=2020/month=08/day=27":</p> <ul style="list-style-type: none"> <li>- If you specify partition root path as "root/folder/year=2020", copy activity will generate two more columns <code>month</code> and <code>day</code> with value "08" and "27" respectively, in addition to the columns inside the files.</li> <li>- If partition root path is not specified, no extra column will be generated.</li> </ul>	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

#### NOTE

For Parquet/delimited text format, the **BlobSource** type for the Copy activity source mentioned in the next section is still supported as is for backward compatibility. We suggest that you use the new model until the authoring UI has switched to generating these new types.

#### Example:

```

"activities": [
 {
 "name": "CopyFromBlob",
 "type": "Copy",
 "inputs": [
 {
 "referenceName": "<Delimited text input dataset name>",
 "type": "DatasetReference"
 }
],
 "outputs": [
 {
 "referenceName": "<output dataset name>",
 "type": "DatasetReference"
 }
],
 "typeProperties": {
 "source": {
 "type": "DelimitedTextSource",
 "formatSettings": {
 "type": "DelimitedTextReadSettings",
 "skipLineCount": 10
 },
 "storeSettings": {
 "type": "AzureBlobStorageReadSettings",
 "recursive": true,
 "wildcardFolderPath": "myfolder*A",
 "wildcardFileName": "*.csv"
 }
 },
 "sink": {
 "type": "<sink type>"
 }
 }
 }
]

```

#### NOTE

The `$logs` container, which is automatically created when Storage Analytics is enabled for a storage account, isn't shown when a container listing operation is performed via the UI. The file path must be provided directly for your data factory or Synapse pipeline to consume files from the `$logs` container.

### Blob storage as a sink type

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- [Avro format](#)
- [Binary format](#)
- [Delimited text format](#)
- [JSON format](#)
- [ORC format](#)
- [Parquet format](#)

The following properties are supported for Azure Blob storage under `storeSettings` settings in a format-based copy sink:

PROPERTY	DESCRIPTION	REQUIRED
----------	-------------	----------

PROPERTY	DESCRIPTION	REQUIRED
type	The <code>type</code> property under <code>storeSettings</code> must be set to <code>AzureBlobStorageWriteSettings</code> .	Yes
copyBehavior	<p>Defines the copy behavior when the source is files from a file-based data store.</p> <p>Allowed values are:</p> <ul style="list-style-type: none"> <li>- <b>PreserveHierarchy (default):</b> Preserves the file hierarchy in the target folder. The relative path of the source file to the source folder is identical to the relative path of the target file to the target folder.</li> <li>- <b>FlattenHierarchy:</b> All files from the source folder are in the first level of the target folder. The target files have autogenerated names.</li> <li>- <b>MergeFiles:</b> Merges all files from the source folder to one file. If the file or blob name is specified, the merged file name is the specified name. Otherwise, it's an autogenerated file name.</li> </ul>	No
blockSizeInMB	<p>Specify the block size, in megabytes, used to write data to block blobs.</p> <p>Learn more <a href="#">about Block Blobs</a>.</p> <p>Allowed value is <i>between 4 MB and 100 MB</i>.</p> <p>By default, the service automatically determines the block size based on your source store type and data. For nonbinary copy into Blob storage, the default block size is 100 MB so it can fit in (at most) 4.95 TB of data. It might be not optimal when your data is not large, especially when you use the self-hosted integration runtime with poor network connections that result in operation timeout or performance issues. You can explicitly specify a block size, while ensuring that <code>blockSizeInMB*50000</code> is big enough to store the data. Otherwise, the Copy activity run will fail.</p>	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

PROPERTY	DESCRIPTION	REQUIRED
metadata	<p>Set custom metadata when copy to sink. Each object under the <code>metadata</code> array represents an extra column. The <code>name</code> defines the metadata key name, and the <code>value</code> indicates the data value of that key. If <a href="#">preserve attributes feature</a> is used, the specified metadata will union/overwrite with the source file metadata.</p> <p>Allowed data values are:</p> <ul style="list-style-type: none"> <li>- <code>\$\$LASTMODIFIED</code> : a reserved variable indicates to store the source files' last modified time. Apply to file-based source with binary format only.</li> <li>- <a href="#">Expression</a></li> <li>- <a href="#">Static value</a></li> </ul>	No

Example:

```

"activities": [
 {
 "name": "CopyFromBlob",
 "type": "Copy",
 "inputs": [
 {
 "referenceName": "<input dataset name>",
 "type": "DatasetReference"
 }
],
 "outputs": [
 {
 "referenceName": "<Parquet output dataset name>",
 "type": "DatasetReference"
 }
],
 "typeProperties": {
 "source": {
 "type": "<source type>"
 },
 "sink": {
 "type": "ParquetSink",
 "storeSettings": {
 "type": "AzureBlobStorageWriteSettings",
 "copyBehavior": "PreserveHierarchy",
 "metadata": [
 {
 "name": "testKey1",
 "value": "value1"
 },
 {
 "name": "testKey2",
 "value": "value2"
 },
 {
 "name": "lastModifiedKey",
 "value": "$$LASTMODIFIED"
 }
]
 }
 }
 }
 }
]

```

## Folder and file filter examples

This section describes the resulting behavior of the folder path and file name with wildcard filters.

FOLDERPATH	FILENAME	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
container/Folder*	(empty, use default)	false	container FolderA <b>File1.csv</b> <b>File2.json</b> Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv

FOLDERPATH	FILENAME	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
container/Folder*	(empty, use default)	true	container FolderA <b>File1.csv</b> <b>File2.json</b> Subfolder1 <b>File3.csv</b> <b>File4.json</b> <b>File5.csv</b> AnotherFolderB <b>File6.csv</b>
container/Folder*	*.csv	false	container FolderA <b>File1.csv</b> <b>File2.json</b> Subfolder1 <b>File3.csv</b> <b>File4.json</b> <b>File5.csv</b> AnotherFolderB <b>File6.csv</b>
container/Folder*	*.csv	true	container FolderA <b>File1.csv</b> <b>File2.json</b> Subfolder1 <b>File3.csv</b> <b>File4.json</b> <b>File5.csv</b> AnotherFolderB <b>File6.csv</b>

### File list examples

This section describes the resulting behavior of using a file list path in the Copy activity source.

Assume that you have the following source folder structure and want to copy the files in bold:

SAMPLE SOURCE STRUCTURE	CONTENT IN FILELISTTOCOPY.TXT	CONFIGURATION
container FolderA <b>File1.csv</b> File2.json Subfolder1 <b>File3.csv</b> File4.json <b>File5.csv</b> Metadata FileListToCopy.txt	File1.csv Subfolder1/File3.csv Subfolder1/File5.csv	<p><b>In dataset:</b></p> <ul style="list-style-type: none"> <li>- Container: <code>container</code></li> <li>- Folder path: <code>FolderA</code></li> </ul> <p><b>In Copy activity source:</b></p> <ul style="list-style-type: none"> <li>- File list path: <code>container/Metadata/FileListToCopy.txt</code></li> </ul> <p>The file list path points to a text file in the same data store that includes a list of files you want to copy, one file per line, with the relative path to the path configured in the dataset.</p>

### Some recursive and copyBehavior examples

This section describes the resulting behavior of the Copy operation for different combinations of **recursive** and

`copyBehavior` values.

RECURSIVE	COPYBEHAVIOR	SOURCE FOLDER STRUCTURE	RESULTING TARGET
true	preserveHierarchy	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target folder, Folder1, is created with the same structure as the source:  Folder1 File1 File2 Subfolder1 File3 File4 File5
true	flattenHierarchy	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target folder, Folder1, is created with the following structure:  Folder1 autogenerated name for File1 autogenerated name for File2 autogenerated name for File3 autogenerated name for File4 autogenerated name for File5
true	mergeFiles	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target folder, Folder1, is created with the following structure:  Folder1 File1 + File2 + File3 + File4 + File5 contents are merged into one file with an autogenerated file name.
false	preserveHierarchy	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target folder, Folder1, is created with the following structure:  Folder1 File1 File2  Subfolder1 with File3, File4, and File5 is not picked up.

Recursive	CopyBehavior	Source Folder Structure	Resulting Target
false	flattenHierarchy	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target folder, Folder1, is created with the following structure:  Folder1 autogenerated name for File1 autogenerated name for File2  Subfolder1 with File3, File4, and File5 is not picked up.
false	mergeFiles	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target folder, Folder1, is created with the following structure:  Folder1 File1 + File2 contents are merged into one file with an autogenerated file name. autogenerated name for File1  Subfolder1 with File3, File4, and File5 is not picked up.

## Preserving metadata during copy

When you copy files from Amazon S3, Azure Blob storage, or Azure Data Lake Storage Gen2 to Azure Data Lake Storage Gen2 or Azure Blob storage, you can choose to preserve the file metadata along with data. Learn more from [Preserve metadata](#).

## Mapping data flow properties

When you're transforming data in mapping data flows, you can read and write files from Azure Blob storage in the following formats:

- [Avro](#)
- [Delimited text](#)
- [Delta](#)
- [Excel](#)
- [JSON](#)
- [Parquet](#)

Format specific settings are located in the documentation for that format. For more information, see [Source transformation in mapping data flow](#) and [Sink transformation in mapping data flow](#).

### Source transformation

In source transformation, you can read from a container, folder, or individual file in Azure Blob storage. Use the [Source options](#) tab to manage how the files are read.

Source settings    Source options **Projection**    Optimize    Inspect    Data preview

---

Wildcard paths	mycontainer / <input type="text" value="partdata/**/*.csv"/> <span style="color: #0078d4;">i</span> <span style="color: #0078d4;">+</span> <span style="color: #0078d4;">-</span>
Partition root path	<input type="text" value="partdata"/> <span style="color: #0078d4;">i</span>
List of files	<input type="checkbox"/> <span style="color: #0078d4;">i</span>
Multiline rows	<input type="checkbox"/> <span style="color: #0078d4;">i</span>
Column to store file name	<input type="text" value="fileName"/> <span style="color: #0078d4;">i</span>
After completion *	<input checked="" type="radio"/> No action <input type="radio"/> Delete source files <input type="radio"/> Move
Start time (UTC)	<input type="text" value="02/01/2020 00:00:00"/>
End time (UTC)	<input type="text" value="02/02/2020 00:00:00"/> <span style="color: #0078d4;">i</span>
Filter by last modified	<input type="text" value=""/> <span style="color: #0078d4;">i</span>

**Wildcard paths:** Using a wildcard pattern will instruct the service to loop through each matching folder and file in a single source transformation. This is an effective way to process multiple files within a single flow. Add multiple wildcard matching patterns with the plus sign that appears when you hover over your existing wildcard pattern.

From your source container, choose a series of files that match a pattern. Only a container can be specified in the dataset. Your wildcard path must therefore also include your folder path from the root folder.

Wildcard examples:

- `*` Represents any set of characters.
- `**` Represents recursive directory nesting.
- `?` Replaces one character.
- `[]` Matches one or more characters in the brackets.
- `/data/sales/**/*.csv` Gets all .csv files under /data/sales.
- `/data/sales/20??/**/` Gets all files in the 20th century.
- `/data/sales/*/*/*.csv` Gets .csv files two levels under /data/sales.
- `/data/sales/2004/*/12/[XY]1?.csv` Gets all .csv files in December 2004 starting with X or Y prefixed by a two-digit number.

**Partition root path:** If you have partitioned folders in your file source with a `key=value` format (for example, `year=2019`), then you can assign the top level of that partition folder tree to a column name in your data flow's data stream.

First, set a wildcard to include all paths that are the partitioned folders plus the leaf files that you want to read.

Wildcard paths mycontainer / partdata/\*\*/\*.\*csv + -

Partition root path partdata + -

List of files + -

Multiline rows + -

Column to store file name myfile + -

After completion \*  No action  Delete source files  Move

Use the **Partition root path** setting to define what the top level of the folder structure is. When you view the contents of your data via a data preview, you'll see that the service will add the resolved partitions found in each of your folder levels.

Projection	Optimize	Inspect	Data Preview	Description
00	* UPDATE 0	X DELETE 0	* UPsert 0	Q LOOKUP 0
				TOTAL 1000
Rating abc	Rotten Tomato abc	releaseyear 123	Month 123	myfile abc
1	54	2019	7	/partdata/releaseyear=2019/...
7	80	2019	7	/partdata/releaseyear=2019/...
6	92	2019	7	/partdata/releaseyear=2019/...
4	82	2019	7	/partdata/releaseyear=2019/...
9	92	2019	7	/partdata/releaseyear=2019/...
4	59	2019	7	/partdata/releaseyear=2019/...
3	83	2019	7	/partdata/releaseyear=2019/...
2	63	2019	7	/partdata/releaseyear=2019/...
4	63	2019	7	/partdata/releaseyear=2019/...
8	50	2019	7	/partdata/releaseyear=2019/...

**List of files:** This is a file set. Create a text file that includes a list of relative path files to process. Point to this text file.

**Column to store file name:** Store the name of the source file in a column in your data. Enter a new column name here to store the file name string.

**After completion:** Choose to do nothing with the source file after the data flow runs, delete the source file, or move the source file. The paths for the move are relative.

To move source files to another location post-processing, first select "Move" for file operation. Then, set the "from" directory. If you're not using any wildcards for your path, then the "from" setting will be the same folder as your source folder.

If you have a source path with wildcard, your syntax will look like this:

```
/data/sales/20??/**/*.csv
```

You can specify "from" as:

```
/data/sales
```

And you can specify "to" as:

```
/backup/priorSales
```

In this case, all files that were sourced under `/data/sales` are moved to `/backup/priorSales`.

#### NOTE

File operations run only when you start the data flow from a pipeline run (a pipeline debug or execution run) that uses the Execute Data Flow activity in a pipeline. File operations *do not* run in Data Flow debug mode.

**Filter by last modified:** You can filter which files you process by specifying a date range of when they were last modified. All datetimes are in UTC.

**Enable change data capture:** If true, you will get new or changed files only from the last run. Initial load of full snapshot data will always be gotten in the first run, followed by capturing new or changed files only in next runs. For more details, see [Change data capture](#).

The screenshot shows the 'Source options' tab of the Azure Data Flow designer. It includes fields for Wildcard paths, Partition root path, Allow no files found, List of files, Multiline rows, and a checkbox for Enable change data capture. The 'Enable change data capture' checkbox is highlighted with a red box. Other settings like Column to store file name and After completion actions are also visible.

#### Sink properties

In the sink transformation, you can write to either a container or a folder in Azure Blob storage. Use the **Settings** tab to manage how the files get written.

The screenshot shows the 'Settings' tab of the Azure Data Flow designer. It includes options for Clear the folder (checked), File name option (Default selected), Pattern, Per partition, As data in column, and Output to single file. There is also a Quote All checkbox.

**Clear the folder:** Determines whether or not the destination folder gets cleared before the data is written.

**File name option:** Determines how the destination files are named in the destination folder. The file name options are:

- **Default:** Allow Spark to name files based on PART defaults.
- **Pattern:** Enter a pattern that enumerates your output files per partition. For example, `loans[n].csv` will create `loans1.csv`, `loans2.csv`, and so on.
- **Per partition:** Enter one file name per partition.
- **As data in column:** Set the output file to the value of a column. The path is relative to the dataset container, not the destination folder. If you have a folder path in your dataset, it will be overridden.
- **Output to a single file:** Combine the partitioned output files into a single named file. The path is relative to the dataset folder. Be aware that the merge operation can possibly fail based on node size. We don't

recommend this option for large datasets.

**Quote all:** Determines whether to enclose all values in quotation marks.

## Lookup activity properties

To learn details about the properties, check [Lookup activity](#).

## GetMetadata activity properties

To learn details about the properties, check [GetMetadata activity](#).

## Delete activity properties

To learn details about the properties, check [Delete activity](#).

## Legacy models

### NOTE

The following models are still supported as is for backward compatibility. We suggest that you use the new model mentioned earlier. The authoring UI has switched to generating the new model.

### Legacy dataset model

PROPERTY	DESCRIPTION	REQUIRED
type	The <code>type</code> property of the dataset must be set to <code>AzureBlob</code> .	Yes
folderPath	<p>Path to the container and folder in Blob storage.</p> <p>A wildcard filter is supported for the path, excluding container name. Allowed wildcards are: <code>*</code> (matches zero or more characters) and <code>?</code> (matches zero or single character). Use <code>^</code> to escape if your folder name has a wildcard or this escape character inside.</p> <p>An example is: <code>myblobcontainer/myblobfolder/</code>. See more examples in <a href="#">Folder and file filter examples</a>.</p>	Yes for the Copy or Lookup activity, No for the GetMetadata activity

PROPERTY	DESCRIPTION	REQUIRED
fileName	<p>Name or wildcard filter for the blobs under the specified <code>FolderPath</code> value. If you don't specify a value for this property, the dataset points to all blobs in the folder.</p> <p>For the filter, allowed wildcards are: <code>*</code> (matches zero or more characters) and <code>?</code> (matches zero or single character).</p> <ul style="list-style-type: none"> <li>- Example 1: <code>"fileName": "*.csv"</code></li> <li>- Example 2: <code>"fileName": "???20180427.txt"</code></li> </ul> <p>Use <code>^</code> to escape if your file name has a wildcard or this escape character inside.</p> <p>When <code>fileName</code> isn't specified for an output dataset and <code>preserveHierarchy</code> isn't specified in the activity sink, the Copy activity automatically generates the blob name with the following pattern: <i>Data.[activity run ID GUID].[GUID if FlattenHierarchy].[format if configured].[compression if configured]</i>. For example: "Data.0a405f8a-93ff-4c6fb3be-f69616f1df7a.txt.gz".</p> <p>If you copy from a tabular source by using a table name instead of a query, the name pattern is <code>[table name].[format].[compression if configured]</code>. For example: "MyTable.csv".</p>	No

PROPERTY	DESCRIPTION	REQUIRED
modifiedDatetimeStart	<p>Files are filtered based on the attribute: last modified. The files will be selected if their last modified time is greater than or equal to <code>modifiedDatetimeStart</code> and less than <code>modifiedDatetimeEnd</code>. The time is applied to the UTC time zone in the format of "2018-12-01T05:00:00Z".</p> <p>Be aware that enabling this setting will affect the overall performance of data movement when you want to filter huge amounts of files.</p> <p>The properties can be <code>NULL</code>, which means no file attribute filter will be applied to the dataset. When <code>modifiedDatetimeStart</code> has a datetime value but <code>modifiedDatetimeEnd</code> is <code>NULL</code>, the files whose last modified attribute is greater than or equal to the datetime value will be selected. When <code>modifiedDatetimeEnd</code> has a datetime value but <code>modifiedDatetimeStart</code> is <code>NULL</code>, the files whose last modified attribute is less than the datetime value will be selected.</p>	No
modifiedDatetimeEnd	<p>Files are filtered based on the attribute: last modified. The files will be selected if their last modified time is greater than or equal to <code>modifiedDatetimeStart</code> and less than <code>modifiedDatetimeEnd</code>. The time is applied to the UTC time zone in the format of "2018-12-01T05:00:00Z".</p> <p>Be aware that enabling this setting will affect the overall performance of data movement when you want to filter huge amounts of files.</p> <p>The properties can be <code>NULL</code>, which means no file attribute filter will be applied to the dataset. When <code>modifiedDatetimeStart</code> has a datetime value but <code>modifiedDatetimeEnd</code> is <code>NULL</code>, the files whose last modified attribute is greater than or equal to the datetime value will be selected. When <code>modifiedDatetimeEnd</code> has a datetime value but <code>modifiedDatetimeStart</code> is <code>NULL</code>, the files whose last modified attribute is less than the datetime value will be selected.</p>	No

PROPERTY	DESCRIPTION	REQUIRED
format	<p>If you want to copy files as is between file-based stores (binary copy), skip the format section in both the input and output dataset definitions.</p> <p>If you want to parse or generate files with a specific format, the following file format types are supported: <b>TextFormat</b>, <b>JsonFormat</b>, <b>AvroFormat</b>, <b>OrcFormat</b>, and <b>ParquetFormat</b>. Set the <b>type</b> property under <b>format</b> to one of these values. For more information, see the <a href="#">Text format</a>, <a href="#">JSON format</a>, <a href="#">Avro format</a>, <a href="#">Orc format</a>, and <a href="#">Parquet format</a> sections.</p>	No (only for binary copy scenario)
compression	<p>Specify the type and level of compression for the data. For more information, see <a href="#">Supported file formats and compression codecs</a>.</p> <p>Supported types are <b>GZip</b>, <b>Deflate</b>, <b>BZip2</b>, and <b>ZipDeflate</b>.</p> <p>Supported levels are <b>Optimal</b> and <b>Fastest</b>.</p>	No

#### TIP

To copy all blobs under a folder, specify **FolderPath** only.

To copy a single blob with a given name, specify **FolderPath** for the folder part and **fileName** for the file name.

To copy a subset of blobs under a folder, specify **FolderPath** for the folder part and **fileName** with a wildcard filter.

**Example:**

```
{
 "name": "AzureBlobDataset",
 "properties": {
 "type": "AzureBlob",
 "linkedServiceName": {
 "referenceName": "<Azure Blob storage linked service name>",
 "type": "LinkedServiceReference"
 },
 "typeProperties": {
 "folderPath": "mycontainer/myfolder",
 "fileName": "*",
 "modifiedDatetimeStart": "2018-12-01T05:00:00Z",
 "modifiedDatetimeEnd": "2018-12-01T06:00:00Z",
 "format": {
 "type": "TextFormat",
 "columnDelimiter": ",",
 "rowDelimiter": "\n"
 },
 "compression": {
 "type": "GZip",
 "level": "Optimal"
 }
 }
 }
}
```

## Legacy source model for the Copy activity

PROPERTY	DESCRIPTION	REQUIRED
type	The <code>type</code> property of the Copy activity source must be set to <code>BlobSource</code> .	Yes
recursive	Indicates whether the data is read recursively from the subfolders or only from the specified folder. Note that when <code>recursive</code> is set to <code>true</code> and the sink is a file-based store, an empty folder or subfolder isn't copied or created at the sink. Allowed values are <code>true</code> (default) and <code>false</code> .	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```

"activities": [
 {
 "name": "CopyFromBlob",
 "type": "Copy",
 "inputs": [
 {
 "referenceName": "<Azure Blob input dataset name>",
 "type": "DatasetReference"
 }
],
 "outputs": [
 {
 "referenceName": "<output dataset name>",
 "type": "DatasetReference"
 }
],
 "typeProperties": {
 "source": {
 "type": "BlobSource",
 "recursive": true
 },
 "sink": {
 "type": "<sink type>"
 }
 }
 }
]

```

## Legacy sink model for the Copy activity

PROPERTY	DESCRIPTION	REQUIRED
type	The <code>type</code> property of the Copy activity sink must be set to <code>BlobSink</code> .	Yes
copyBehavior	<p>Defines the copy behavior when the source is files from a file-based data store.</p> <p>Allowed values are:</p> <ul style="list-style-type: none"> <li>- <b>PreserveHierarchy (default):</b> Preserves the file hierarchy in the target folder. The relative path of source file to source folder is identical to the relative path of target file to target folder.</li> <li>- <b>FlattenHierarchy:</b> All files from the source folder are in the first level of the target folder. The target files have autogenerated names.</li> <li>- <b>MergeFiles:</b> Merges all files from the source folder to one file. If the file or blob name is specified, the merged file name is the specified name. Otherwise, it's an autogenerated file name.</li> </ul>	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

## Example:

```
"activities": [
 {
 "name": "CopyToBlob",
 "type": "Copy",
 "inputs": [
 {
 "referenceName": "<input dataset name>",
 "type": "DatasetReference"
 }
],
 "outputs": [
 {
 "referenceName": "<Azure Blob output dataset name>",
 "type": "DatasetReference"
 }
],
 "typeProperties": {
 "source": {
 "type": "<source type>"
 },
 "sink": {
 "type": "BlobSink",
 "copyBehavior": "PreserveHierarchy"
 }
 }
 }
]
```

## Change data capture (preview)

Azure Data Factory can get new or changed files only from Azure Blob Storage by enabling **Enable change data capture (Preview)** in the mapping data flow source transformation. With this connector option, you can read new or updated files only and apply transformations before loading transformed data into destination datasets of your choice.

Make sure you keep the pipeline and activity name unchanged, so that the checkpoint can always be recorded from the last run to get changes from there. If you change your pipeline name or activity name, the checkpoint will be reset, and you will start from the beginning in the next run.

When you debug the pipeline, the **Enable change data capture (Preview)** works as well. Be aware that the checkpoint will be reset when you refresh your browser during the debug run. After you are satisfied with the result from debug run, you can publish and trigger the pipeline. It will always start from the beginning regardless of the previous checkpoint recorded by debug run.

In the monitoring section, you always have the chance to rerun a pipeline. When you are doing so, the changes are always gotten from the checkpoint record in your selected pipeline run.

## Next steps

For a list of data stores that the Copy activity supports as sources and sinks, see [Supported data stores](#).

# Connect to Azure Blob Storage by using the SSH File Transfer Protocol (SFTP) (preview)

8/22/2022 • 13 minutes to read • [Edit Online](#)

You can securely connect to the Blob Storage endpoint of an Azure Storage account by using an SFTP client, and then upload and download files. This article shows you how to enable SFTP, and then connect to Blob Storage by using an SFTP client.

To learn more about SFTP support for Azure Blob Storage, see [SSH File Transfer Protocol \(SFTP\) in Azure Blob Storage](#).

## IMPORTANT

SFTP support is currently in PREVIEW and is available on general-purpose v2 and premium block blob accounts. Complete [this form](#) BEFORE using the feature in preview. Registration via 'preview features' is NOT required and confirmation email will NOT be sent after filling out the form. You can IMMEDIATELY access the feature.

After testing your end-to-end scenarios with SFTP, please share your experience via [this form](#).

See the [Supplemental Terms of Use for Microsoft Azure Previews](#) for legal terms that apply to Azure features that are in beta, preview, or otherwise not yet released into general availability.

## Prerequisites

- A standard general-purpose v2 or premium block blob storage account. You can also enable SFTP as you create the account. For more information on these types of storage accounts, see [Storage account overview](#).
- The hierarchical namespace feature of the account must be enabled. To enable the hierarchical namespace feature, see [Upgrade Azure Blob Storage with Azure Data Lake Storage Gen2 capabilities](#).
- If you're connecting from an on-premises network, make sure that your client allows outgoing communication through port 22 used by SFTP.

## Enable SFTP support

This section shows you how to enable SFTP support for an existing storage account. To view an Azure Resource Manager template that enables SFTP support as part of creating the account, see [Create an Azure Storage Account and Blob Container accessible using SFTP protocol on Azure](#). To view the Local User REST APIs and .NET references, see [Local Users](#) and [LocalUser Class](#).

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

1. In the [Azure portal](#), navigate to your storage account.
2. Under **Settings**, select **SFTP**.

#### NOTE

This option appears only if the hierarchical namespace feature of the account has been enabled. To enable the hierarchical namespace feature, see [Upgrade Azure Blob Storage with Azure Data Lake Storage Gen2 capabilities](#).

### 3. Select Enable SFTP.

The screenshot shows the Azure Storage account settings for 'contoso'. The 'SFTP' tab is selected in the left sidebar. The 'Enable SFTP' checkbox is checked and highlighted with a red box. A note below it says 'SFTP is disabled for this account. In order to be able to SFTP into accounts, enable this feature to start using it again. You can still add/edit local users.' A 'Create or edit local users below in order to utilize Secure File Transfer Protocol' message is also present.

#### NOTE

If no local users appear in the SFTP configuration page, you'll need to add at least one of them. To add local users, see the next section.

## Configure permissions

Azure Storage doesn't support shared access signature (SAS), or Azure Active directory (Azure AD) authentication for accessing the SFTP endpoint. Instead, you must use an identity called local user that can be secured with an Azure generated password or a secure shell (SSH) key pair. To grant access to a connecting client, the storage account must have an identity associated with the password or key pair. That identity is called a *local user*.

In this section, you'll learn how to create a local user, choose an authentication method, and assign permissions for that local user.

To learn more about the SFTP permissions model, see [SFTP Permissions model](#).

#### TIP

This section shows you how to configure local users for an existing storage account. To view an Azure Resource Manager template that configures a local user as part of creating an account, see [Create an Azure Storage Account and Blob Container accessible using SFTP protocol on Azure](#).

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

1. In the [Azure portal](#), navigate to your storage account.
2. Under **Settings**, select **SFTP**, and then select **Add local user**.

The screenshot shows the Microsoft Azure portal interface for a storage account named 'contoso'. The left sidebar has a tree view with 'Settings' expanded, and 'SFTP' is selected and highlighted with a red box. The main content area shows a table for managing local users. At the top right of the table, there's a 'Create or edit local users below in order to utilize S' message. Below the table, a note says 'No local users found. Add a local user'.

3. In the **Add local user** configuration pane, add the name of a user, and then select which methods of authentication you'd like associate with this local user. You can associate a password and / or an SSH key.

#### IMPORTANT

While you can enable both forms of authentication, SFTP clients can connect by using only one of them. Multifactor authentication, whereby both a valid password and a valid public and private key pair are required for successful authentication is not supported.

If you select **SSH Password**, then your password will appear when you've completed all of the steps in the **Add local user** configuration pane. Note that SSH passwords are generated by Azure and are minimum 88 characters in length.

If you select **SSH Key pair**, then select **Public key source** to specify a key source.

# Add local user

X

## Username + Authentication

## Container permissions

Username \*

contoso

### Authentication methods

Authentication methods allowed for this user

SSH Password



 You'll be able to retrieve your password once the local user has been added.

SSH Key pair



### SSH keys

 Add key

### Public key source

### Key name OR Public key

Generate new key pair



Generate new key pair

Use existing key stored in Azure

Use existing public key



The following table describes each key source option:

OPTION	GUIDANCE
Generate a new key pair	Use this option to create a new public / private key pair. The public key is stored in Azure with the key name that you provide. The private key can be downloaded after the local user has been successfully added.
Use existing key stored in Azure	Use this option if you want to use a public key that is already stored in Azure. To find existing keys in Azure, see <a href="#">List keys</a> . When SFTP clients connect to Azure Blob Storage, those clients need to provide the private key associated with this public key.
Use existing public key	Use this option if you want to upload a public key that is stored outside of Azure. If you don't have a public key, but would like to generate one outside of Azure, see <a href="#">Generate keys with ssh-keygen</a> .

4. Select **Next** to open the **Container permissions** tab of the configuration pane.
5. In the **Container permissions** tab, select the containers that you want to make available to this local user. Then, select which types of operations you want to enable this local user to perform.

## Container permissions

Decide which containers this local user has access to.

Containers

contosocontainer ▼  
[Create new](#)

Container name	Permissions
contosocontainer	<input type="checkbox"/> Read <input checked="" type="checkbox"/> Write <input type="checkbox"/> List <input type="checkbox"/> Delete <input type="checkbox"/> Create
Home directory ⓘ	

6. In the **Home directory** edit box, type the name of the container or the directory path (including the container name) that will be the default location associated with this local user.

To learn more about the home directory, see [Home directory](#).

7. Select the **Add button** to add the local user.

If you enabled password authentication, then the Azure generated password appears in a dialog box after the local user has been added.

### IMPORTANT

You can't retrieve this password later, so make sure to copy the password, and then store it in a place where you can find it.

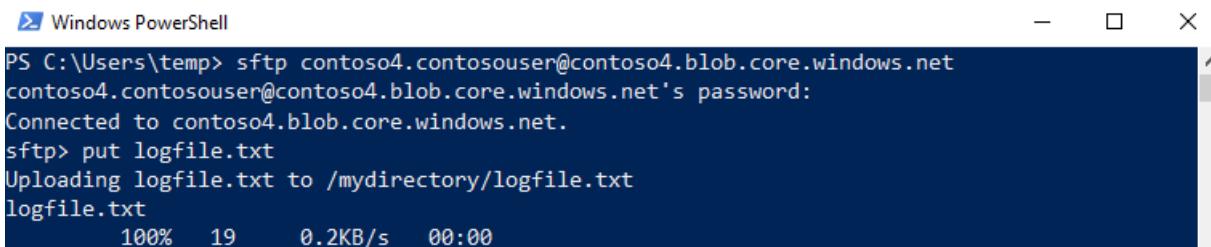
If you chose to generate a new key pair, then you'll be prompted to download the private key of that key pair after the local user has been added.

### NOTE

Local users have a `sharedKey` property that is used for SMB authentication only.

## Connect an SFTP client

You can use any SFTP client to securely connect and then transfer files. The following screenshot shows a Windows PowerShell session that uses [Open SSH](#) and password authentication to connect and then upload a file named `logfile.txt`.



```
PS C:\Users\temp> sftp contoso4.contosouser@contoso4.blob.core.windows.net
contoso4.contosouser@contoso4.blob.core.windows.net's password:
Connected to contoso4.blob.core.windows.net.
sftp> put logfile.txt
Uploading logfile.txt to /mydirectory/logfile.txt
logfile.txt
 100% 19 0.2KB/s 00:00
```

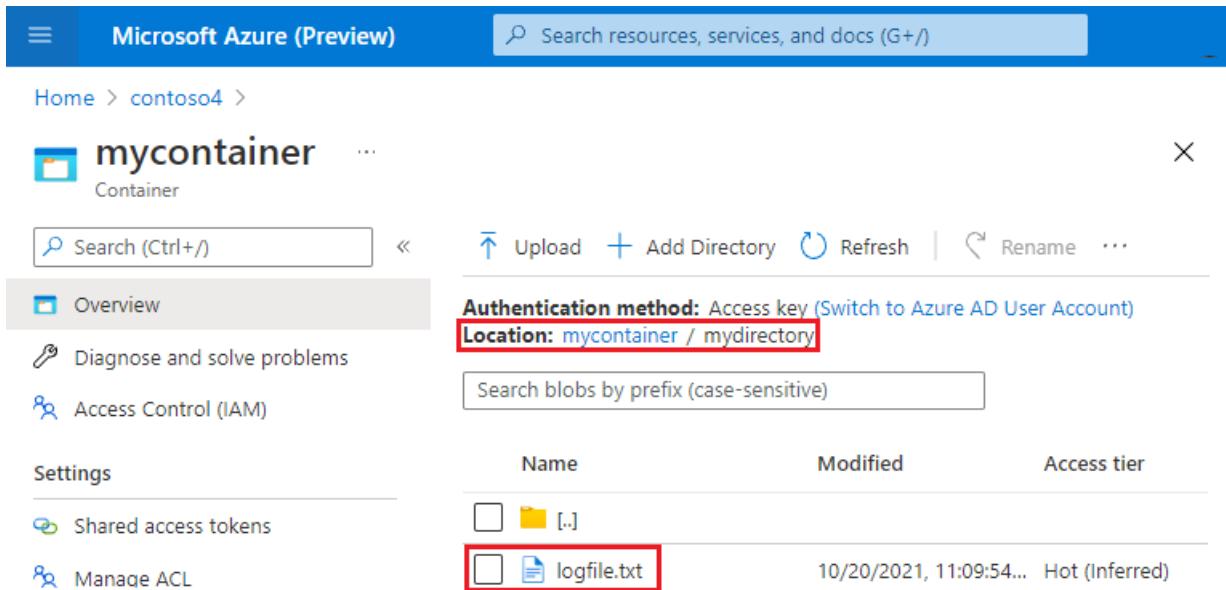
#### NOTE

The SFTP username is `storage_account_name`.`username`. In the example above the `storage_account_name` is "contoso4" and the `username` is "contosouser." The combined username becomes `contoso4.contosouser` for the SFTP command.

#### NOTE

You might be prompted to trust a host key. During the public preview, valid host keys are published [here](#).

After the transfer is complete, you can view and manage the file in the Azure portal.



The screenshot shows the Azure portal interface for a blob storage container named 'mycontainer'. The left sidebar has options like Overview, Diagnose and solve problems, and Access Control (IAM). The main area shows the container details with an 'Upload' button, 'Add Directory' button, 'Refresh' button, and 'Rename' button. A search bar at the top says 'Search resources, services, and docs (G+ /)'. Below the search bar, it says 'Authentication method: Access key (Switch to Azure AD User Account)' and 'Location: mycontainer / mydirectory'. A search bar below that says 'Search blobs by prefix (case-sensitive)'. The file list table has columns for Name, Modified, and Access tier. It shows one file, 'logfile.txt', with a checkbox next to it. The 'Modified' column shows '10/20/2021, 11:09:54... Hot (Inferred)'.

#### NOTE

The Azure portal uses the Blob REST API and Data Lake Storage Gen2 REST API. Being able to interact with an uploaded file in the Azure portal demonstrates the interoperability between SFTP and REST.

See the documentation of your SFTP client for guidance about how to connect and transfer files.

## Connect using a custom domain

When using custom domains the connection string is `myaccount.myuser@customdomain.com`. If home directory has not been specified for the user, it is `myaccount.mycontainer.myuser@customdomain.com`.

#### IMPORTANT

Ensure your DNS provider does not proxy requests. Proxying may cause the connection attempt to time out.

## Connect using a private endpoint

When using a private endpoint the connection string is

`myaccount.myuser@myaccount.privatelink.blob.core.windows.net`. If home directory has not been specified for the user, it is `myaccount.mycontainer.myuser@myaccount.privatelink.blob.core.windows.net`.

**NOTE**

Ensure you change networking configuration to "Enabled from selected virtual networks and IP addresses" and select your private endpoint, otherwise the regular SFTP endpoint will still be publicly accessible.

## Networking considerations

SFTP is a platform level service, so port 22 will be open even if the account option is disabled. If SFTP access is not configured then all requests will receive a disconnect from the service. When using SFTP may want to limit public access through configuration of a firewall, virtual network, or private endpoint. These settings are enforced at the application layer, which means they are not specific to SFTP and will impact connectivity to all Azure Storage Endpoints. For more information on firewalls and network configuration, see [Configure Azure Storage firewalls and virtual networks](#).

**NOTE**

Audit tools that attempt to determine TLS support at the protocol layer may return TLS versions in addition to the minimum required version when run directly against the storage account endpoint. For more information, see [Enforce a minimum required version of Transport Layer Security \(TLS\) for requests to a storage account](#).

## See also

- [SSH File Transfer Protocol \(SFTP\) support for Azure Blob Storage](#)
- [Limitations and known issues with SSH File Transfer Protocol \(SFTP\) support for Azure Blob Storage](#)
- [Host keys for SSH File Transfer Protocol \(SFTP\) support for Azure Blob Storage](#)
- [SSH File Transfer Protocol \(SFTP\) performance considerations in Azure Blob storage](#)

# Mount Blob Storage by using the Network File System (NFS) 3.0 protocol

8/22/2022 • 4 minutes to read • [Edit Online](#)

This article provides guidance on how to mount a container in Azure Blob Storage from a Linux-based Azure virtual machine (VM) or a Linux system that runs on-premises by using the Network File System (NFS) 3.0 protocol. To learn more about NFS 3.0 protocol support in Blob Storage, see [Network File System \(NFS\) 3.0 protocol support for Azure Blob Storage](#).

## Step 1: Create an Azure virtual network

Your storage account must be contained within a virtual network. A virtual network enables clients to connect securely to your storage account. To learn more about Azure Virtual Network, and how to create a virtual network, see the [Virtual Network documentation](#).

### NOTE

Clients in the same virtual network can mount containers in your account. You can also mount a container from a client that runs in an on-premises network, but you'll have to first connect your on-premises network to your virtual network. See [Supported network connections](#).

## Step 2: Configure network security

Currently, the only way to secure the data in your storage account is by using a virtual network and other network security settings. Any other tools used to secure data, including account key authorization, Azure Active Directory (Azure AD) security, and access control lists (ACLs), are not yet supported in accounts that have the NFS 3.0 protocol support enabled on them.

To secure the data in your account, see these recommendations: [Network security recommendations for Blob storage](#).

## Step 3: Create and configure a storage account

To mount a container by using NFS 3.0, you must create a storage account. You can't enable existing accounts.

The NFS 3.0 protocol is supported for standard general-purpose v2 storage accounts and for premium block blob storage accounts. For more information on these types of storage accounts, see [Storage account overview](#).

To configure the account, choose these values:

SETTING	PREMIUM PERFORMANCE	STANDARD PERFORMANCE
Location	All available regions	All available regions
Performance	Premium	Standard
Account kind	BlockBlobStorage	General-purpose V2

SETTING	PREMIUM PERFORMANCE	STANDARD PERFORMANCE
Replication	Locally-redundant storage (LRS), Zone-redundant storage (ZRS)	Locally-redundant storage (LRS), Zone-redundant storage (ZRS)
Connectivity method	Public endpoint (selected networks) or Private endpoint	Public endpoint (selected networks) or Private endpoint
Hierarchical namespace	Enabled	Enabled
NFS V3	Enabled	Enabled

You can accept the default values for all other settings.

## Step 4: Create a container

Create a container in your storage account by using any of these tools or SDKs:

TOOLS	SDKS
<a href="#">Azure portal</a>	<a href="#">.NET</a>
<a href="#">AzCopy</a>	<a href="#">Java</a>
<a href="#">PowerShell</a>	<a href="#">Python</a>
<a href="#">Azure CLI</a>	<a href="#">JavaScript</a>
	<a href="#">REST</a>

### NOTE

By default, the root squash option of a new container is **No Root Squash**. But you can change that to **Root Squash** or **All Squash**. For information about these squash options, see your operating system documentation.

The following image shows the squash options as they appear in the Azure portal.

## New container

X

Name \*

mynewcontainer ✓

Public access level ⓘ

Private (no anonymous access) ▾

Advanced

Root Squash \* ⓘ

No Root Squash ▾

No Root Squash

Root Squash

All Squash

## Step 5: Mount the container

Create a directory on your Linux system, and then mount the container in the storage account.

1. On your Linux system, create a directory:

```
mkdir -p /nfsdata
```

2. Mount the container by using one of the following methods. In both methods, replace the

<storage-account-name> placeholder with the name of your storage account, and replace

<container-name> with the name of your container.

- To have the share mounted automatically on reboot:

- a. Create an entry in the /etc/fstab file by adding the following line:

```
<storage-account-name>.blob.core.windows.net:<storage-account-name>/<container-name>
/nfsdata nfs defaults,sec=sys,vers=3,nolock,proto=tcp,nofail 0 0
```

- b. Run the following command to immediately process the /etc/fstab entries and attempt to mount the preceding path:

```
mount /nfsdata
```

- For a temporary mount that doesn't persist across reboots, run the following command:

```
mount -o sec=sys,vers=3,nolock,proto=tcp <storage-account-
name>.blob.core.windows.net:<storage-account-name>/<container-name> /nfsdata
```

## Resolve common errors

ERROR

CAUSE/RESOLUTION

ERROR	CAUSE/RESOLUTION
<code>Access denied by server while mounting</code>	Ensure that your client is running within a supported subnet. See <a href="#">Supported network locations</a> .
<code>No such file or directory</code>	Make sure to type, rather than copy and paste, the mount command and its parameters directly into the terminal. If you copy and paste any part of this command into the terminal from another application, hidden characters in the pasted information might cause this error to appear. This error also might appear if the account isn't enabled for NFS 3.0.
<code>Permission denied</code>	The default mode of a newly created NFS 3.0 container is 0750. Non-root users don't have access to the volume. If access from non-root users is required, root users must change the mode to 0755. Sample command: <code>sudo chmod 0755 /nfsdata</code>
<code>EINVAL ("Invalid argument" )</code>	This error can appear when a client attempts to: <ul style="list-style-type: none"><li>• Write to a blob that was created from a blob endpoint.</li><li>• Delete a blob that has a snapshot or is in a container that has an active WORM (write once, read many) policy.</li></ul>
<code>EROFS ("Read-only file system" )</code>	This error can appear when a client attempts to: <ul style="list-style-type: none"><li>• Write to a blob or delete a blob that has an active lease.</li><li>• Write to a blob or delete a blob in a container that has an active WORM policy.</li></ul>
<code>NFS3ERR_IO/EIO ("Input/output error" )</code>	This error can appear when a client attempts to read, write, or set attributes on blobs that are stored in the archive access tier.
<code>OperationNotSupportedOnSymLink error</code>	This error can be returned during a write operation via a Blob Storage or Azure Data Lake Storage Gen2 API. Using these APIs to write or delete symbolic links that are created by using NFS 3.0 is not allowed. Make sure to use the NFS 3.0 endpoint to work with symbolic links.
<code>mount: /nfsdata: bad option;</code>	Install the NFS helper program by using <code>sudo apt install nfs-common</code> .

## See also

- [Network File System \(NFS\) 3.0 protocol support for Azure Blob Storage](#)
- [Known issues with Network File System \(NFS\) 3.0 protocol support for Azure Blob Storage](#)

# How to mount Blob storage as a file system with BlobFuse

8/22/2022 • 4 minutes to read • [Edit Online](#)

## Overview

### NOTE

This article is about the original version of BlobFuse. It is simply referred to as "BlobFuse" in many cases, but is also referred to as "BlobFuse v1" in this and other articles to distinguish it from the next generation of BlobFuse, BlobFuse2. BlobFuse2 is currently in preview and might not be suitable for production workloads.

To learn about the improvements made in BlobFuse2, see [What is BlobFuse2?](#).

**BlobFuse** is a virtual file system driver for Azure Blob storage. BlobFuse allows you to access your existing block blob data in your storage account through the Linux file system. BlobFuse uses the virtual directory scheme with the forward-slash '/' as a delimiter.

This guide shows you how to use BlobFuse, and mount a Blob storage container on Linux and access data. To learn more about BlobFuse, see the [readme](#) and [wiki](#).

### WARNING

BlobFuse doesn't guarantee 100% POSIX compliance as it simply translates requests into [Blob REST APIs](#). For example, rename operations are atomic in POSIX, but not in BlobFuse. For a full list of differences between a native file system and BlobFuse, visit [the BlobFuse source code repository](#).

## Install BlobFuse on Linux

BlobFuse binaries are available on [the Microsoft software repositories for Linux](#) for Ubuntu, Debian, SUSE, CentOS, Oracle Linux and RHEL distributions. To install BlobFuse on those distributions, configure one of the repositories from the list. You can also build the binaries from source code following the [Azure Storage installation steps](#) if there are no binaries available for your distribution.

BlobFuse is published in the Linux repo for Ubuntu versions: 16.04, 18.04, and 20.04, RHEL versions: 7.5, 7.8, 7.9, 8.0, 8.1, 8.2, CentOS versions: 7.0, 8.0, Debian versions: 9.0, 10.0, SUSE version: 15, OracleLinux 8.1 . Run this command to make sure that you have one of those versions deployed:

```
lsb_release -a
```

### Configure the Microsoft package repository

Configure the [Linux Package Repository for Microsoft Products](#).

As an example, on an Enterprise Linux 8 distribution:

```
sudo rpm -Uvh https://packages.microsoft.com/config/rhel/8/packages-microsoft-prod.rpm
```

Similarly, change the URL to `.../rhel/7/...` to point to an Enterprise Linux 7 distribution.

Another example on an Ubuntu 20.04 distribution:

```
wget https://packages.microsoft.com/config/ubuntu/20.04/packages-microsoft-prod.deb
sudo dpkg -i packages-microsoft-prod.deb
sudo apt-get update
```

Similarly, change the URL to `.../ubuntu/16.04/...` or `.../ubuntu/18.04/...` to reference another Ubuntu version.

## Install BlobFuse

On an Ubuntu/Debian distribution:

```
sudo apt-get install blobfuse
```

On an Enterprise Linux distribution:

```
sudo yum install blobfuse
```

On a SUSE distribution:

```
sudo zypper install blobfuse
```

## Prepare for mounting

BlobFuse provides native-like performance by requiring a temporary path in the file system to buffer and cache any open files. For this temporary path, choose the most performant disk, or use a ramdisk for best performance.

### NOTE

BlobFuse stores all open file contents in the temporary path. Make sure to have enough space to accommodate all open files.

### (Optional) Use a ramdisk for the temporary path

The following example creates a ramdisk of 16 GB and a directory for BlobFuse. Choose the size based on your needs. This ramdisk allows BlobFuse to open files up to 16 GB in size.

```
sudo mkdir /mnt/ramdisk
sudo mount -t tmpfs -o size=16g tmpfs /mnt/ramdisk
sudo mkdir /mnt/ramdisk/blobfusetmp
sudo chown <youruser> /mnt/ramdisk/blobfusetmp
```

### Use an SSD as a temporary path

In Azure, you may use the ephemeral disks (SSD) available on your VMs to provide a low-latency buffer for BlobFuse. Depending on the provisioning agent used, the ephemeral disk would be mounted on '/mnt' for cloud-init or '/mnt/resource' for waagent VMs.

Make sure your user has access to the temporary path:

```
sudo mkdir /mnt/resource/blobfusetmp -p
sudo chown <youruser> /mnt/resource/blobfusetmp
```

## Authorize access to your storage account

You can authorize access to your storage account by using the account access key, a shared access signature, a managed identity, or a service principal. Authorization information can be provided on the command line, in a config file, or in environment variables. For details, see [Valid authentication setups](#) in the BlobFuse readme.

For example, suppose you are authorizing with the account access keys and storing them in a config file. The config file should have the following format:

```
accountName myaccount
accountKey storageaccesskey
containerName mycontainer
```

The `accountName` is the name of your storage account, and not the full URL.

Create this file using:

```
touch /path/to/fuse_connection.cfg
```

Once you've created and edited this file, make sure to restrict access so no other users can read it.

```
chmod 600 /path/to/fuse_connection.cfg
```

### NOTE

If you have created the configuration file on Windows, make sure to run `dos2unix` to sanitize and convert the file to Unix format.

## Create an empty directory for mounting

```
mkdir ~/mycontainer
```

## Mount

### NOTE

For a full list of mount options, check [the BlobFuse repository](#).

To mount BlobFuse, run the following command with your user. This command mounts the container specified in '/path/to/fuse\_connection.cfg' onto the location '/mycontainer'.

```
blobfuse ~/mycontainer --tmp-path=/mnt/resource/blobfusetmp --config-file=/path/to/fuse_connection.cfg -o
attr_timeout=240 -o entry_timeout=240 -o negative_timeout=120
```

#### NOTE

If you use an ADLS account, you must include `--use-adls=true`.

You should now have access to your block blobs through the regular file system APIs. The user who mounts the directory is the only person who can access it, by default, which secures the access. To allow access to all users, you can mount via the option `-o allow_other`.

```
cd ~/mycontainer
mkdir test
echo "hello world" > test/blob.txt
```

## Persist the mount

To learn how to persist the mount, see [Persisting](#) in the BlobFuse wiki.

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## Next steps

- [BlobFuse home page](#)
- [Report BlobFuse issues](#)

# Transfer data with the Data Movement library

8/22/2022 • 11 minutes to read • [Edit Online](#)

The Azure Storage Data Movement library is a cross-platform open source library that is designed for high performance uploading, downloading, and copying of blobs and files. The Data Movement library provides convenient methods that aren't available in the Azure Storage client library for .NET. These methods provide the ability to set the number of parallel operations, track transfer progress, easily resume a canceled transfer, and much more.

This library also uses .NET Core, which means you can use it when building .NET apps for Windows, Linux and macOS. To learn more about .NET Core, refer to the [.NET Core documentation](#). This library also works for traditional .NET Framework apps for Windows.

This document demonstrates how to create a .NET Core console application that runs on Windows, Linux, and macOS and performs the following scenarios:

- Upload files and directories to Blob Storage.
- Define the number of parallel operations when transferring data.
- Track data transfer progress.
- Resume canceled data transfer.
- Copy file from URL to Blob Storage.
- Copy from Blob Storage to Blob Storage.

## Prerequisites

- [Visual Studio Code](#)
- An [Azure storage account](#)

## Setup

1. Visit the [.NET Core Installation Guide](#) to install the .NET Core SDK. When selecting your environment, choose the command-line option.
2. From the command line, create a directory for your project. Navigate into this directory, then type `dotnet new console -o <sample-project-name>` to create a C# console project.
3. Open this directory in Visual Studio Code. This step can be quickly done via the command line by typing `code .` in Windows.
4. Install the [C# extension](#) from the Visual Studio Code Marketplace. Restart Visual Studio Code.
5. At this point, you should see two prompts. One is for adding "required assets to build and debug." Click "yes." Another prompt is for restoring unresolved dependencies. Click "restore."
6. Modify `launch.json` under `.vscode` to use external terminal as a console. This setting should read as`"console": "externalTerminal"`
7. Visual Studio Code allows you to debug .NET Core applications. Hit `F5` to run your application and verify that your setup is working. You should see "Hello World!" printed to the console.

## Add the Data Movement library to your project

1. Add the latest version of the Data Movement library to the `dependencies` section of your `<project-name>.csproj` file. At the time of writing, this version would be

```
"Microsoft.Azure.Storage.DataMovement": "0.6.2"
```

2. A prompt should display to restore your project. Click the "restore" button. You can also restore your project from the command line by typing the command `dotnet restore` in the root of your project directory.

Modify `<project-name>.csproj` :

```
<Project Sdk="Microsoft.NET.Sdk">
 <PropertyGroup>
 <OutputType>Exe</OutputType>
 <TargetFramework>netcoreapp2.0</TargetFramework>
 </PropertyGroup>
 <ItemGroup>
 <PackageReference Include="Microsoft.Azure.Storage.DataMovement" Version="0.6.2" />
 </ItemGroup>
</Project>
```

## Set up the skeleton of your application

The first thing we do is set up the "skeleton" code of our application. This code prompts us for a Storage account name and account key and uses those credentials to create a `CloudStorageAccount` object. This object is used to interact with our Storage account in all transfer scenarios. The code also prompts us to choose the type of transfer operation we would like to execute.

Modify `Program.cs` :

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Diagnostics;
using Microsoft.Azure.Storage;
using Microsoft.Azure.Storage.Blob;
using Microsoft.Azure.Storage.DataMovement;

namespace DMLibSample
{
 public class Program
 {
 public static void Main()
 {
 Console.WriteLine("Enter Storage account name:");
 string accountName = Console.ReadLine();

 Console.WriteLine("\nEnter Storage account key:");
 string accountKey = Console.ReadLine();

 string storageConnectionString = "DefaultEndpointsProtocol=https;AccountName=" + accountName +
";AccountKey=" + accountKey;
 CloudStorageAccount account = CloudStorageAccount.Parse(storageConnectionString);

 ExecuteChoice(account);
 }

 public static void ExecuteChoice(CloudStorageAccount account)
 {
 Console.WriteLine("\nWhat type of transfer would you like to execute?\n1. Local file --> Azure
Blob\n2. Local directory --> Azure Blob directory\n3. URL (e.g. Amazon S3 file) --> Azure Blob\n4. Azure
Blob --> Azure Blob");
 int choice = int.Parse(Console.ReadLine());

 if(choice == 1)
 {
 TransferLocalFileToAzureBlob(account).Wait();
 }
 }
 }
}
```

```
 else if(choice == 2)
 {
 TransferLocalDirectoryToAzureBlobDirectory(account).Wait();
 }
 else if(choice == 3)
 {
 TransferUrlToAzureBlob(account).Wait();
 }
 else if(choice == 4)
 {
 TransferAzureBlobToAzureBlob(account).Wait();
 }
}

public static async Task TransferLocalFileToAzureBlob(CloudStorageAccount account)
{
}

public static async Task TransferLocalDirectoryToAzureBlobDirectory(CloudStorageAccount account)
{
}

public static async Task TransferUrlToAzureBlob(CloudStorageAccount account)
{
}

public static async Task TransferAzureBlobToAzureBlob(CloudStorageAccount account)
{
}
}
```

## Upload a local file to a blob

Add the methods `GetSourcePath` and `GetBlob` to `Program.cs`:

```
public static string GetSourcePath()
{
 Console.WriteLine("\nProvide path for source:");
 string sourcePath = Console.ReadLine();

 return sourcePath;
}

public static CloudBlockBlob GetBlob(CloudStorageAccount account)
{
 CloudBlobClient blobClient = account.CreateCloudBlobClient();

 Console.WriteLine("\nProvide name of Blob container:");
 string containerName = Console.ReadLine();
 CloudBlobContainer container = blobClient.GetContainerReference(containerName);
 container.CreateIfNotExistsAsync().Wait();

 Console.WriteLine("\nProvide name of new Blob:");
 string blobName = Console.ReadLine();
 CloudBlockBlob blob = container.GetBlockBlobReference(blobName);

 return blob;
}
```

Modify the `TransferLocalFileToAzureBlob` method:

```
public static async Task TransferLocalFileToAzureBlob(CloudStorageAccount account)
{
 string localFilePath = GetSourcePath();
 CloudBlockBlob blob = GetBlob(account);
 Console.WriteLine("\nTransfer started...");
 await TransferManager.UploadAsync(localFilePath, blob);
 Console.WriteLine("\nTransfer operation complete.");
 ExecuteChoice(account);
}
```

This code prompts us for the path to a local file, the name of a new or existing container, and the name of a new blob. The `TransferManager.UploadAsync` method performs the upload using this information.

Hit `F5` to run your application. You can verify that the upload occurred by viewing your Storage account with the [Microsoft Azure Storage Explorer](#).

## Set the number of parallel operations

One feature offered by the Data Movement library is the ability to set the number of parallel operations to increase the data transfer throughput. By default, the Data Movement library sets the number of parallel operations to  $8 * \text{the number of cores on your machine}$ .

Keep in mind that many parallel operations in a low-bandwidth environment may overwhelm the network connection and actually prevent operations from fully completing. You'll need to experiment with this setting to determine what works best based on your available network bandwidth.

Let's add some code that allows us to set the number of parallel operations. Let's also add code that times how long it takes for the transfer to complete.

Add a `SetNumberOfParallelOperations` method to `Program.cs`:

```
public static void SetNumberOfParallelOperations()
{
 Console.WriteLine("\nHow many parallel operations would you like to use?");
 string parallelOperations = Console.ReadLine();
 TransferManager.Configurations.ParallelOperations = int.Parse(parallelOperations);
}
```

Modify the `ExecuteChoice` method to use `SetNumberOfParallelOperations`:

```

public static void ExecuteChoice(CloudStorageAccount account)
{
 Console.WriteLine("\nWhat type of transfer would you like to execute?\n1. Local file --> Azure Blob\n2.
Local directory --> Azure Blob directory\n3. URL (e.g. Amazon S3 file) --> Azure Blob\n4. Azure Blob -->
Azure Blob");
 int choice = int.Parse(Console.ReadLine());

 SetNumberOfParallelOperations();

 if(choice == 1)
 {
 TransferLocalFileToAzureBlob(account).Wait();
 }
 else if(choice == 2)
 {
 TransferLocalDirectoryToAzureBlobDirectory(account).Wait();
 }
 else if(choice == 3)
 {
 TransferUrlToAzureBlob(account).Wait();
 }
 else if(choice == 4)
 {
 TransferAzureBlobToAzureBlob(account).Wait();
 }
}

```

Modify the `TransferLocalFileToAzureBlob` method to use a timer:

```

public static async Task TransferLocalFileToAzureBlob(CloudStorageAccount account)
{
 string localFilePath = GetSourcePath();
 CloudBlockBlob blob = GetBlob(account);
 Console.WriteLine("\nTransfer started...");
 Stopwatch stopWatch = Stopwatch.StartNew();
 await TransferManager.UploadAsync(localFilePath, blob);
 stopWatch.Stop();
 Console.WriteLine("\nTransfer operation completed in " + stopWatch.Elapsed.TotalSeconds + " seconds.");
 ExecuteChoice(account);
}

```

## Track transfer progress

Knowing how long it took for the data to transfer is helpful. However, being able to see the progress of the transfer *during* the transfer operation would be even better. To achieve this scenario, we need to create a `TransferContext` object. The `TransferContext` object comes in two forms: `SingleTransferContext` and `DirectoryTransferContext`. The former is for transferring a single file and the latter is for transferring a directory of files.

Add the methods `GetSingleTransferContext` and `GetDirectoryTransferContext` to `Program.cs`:

```

public static SingleTransferContext GetSingleTransferContext(TransferCheckpoint checkpoint)
{
 SingleTransferContext context = new SingleTransferContext(checkpoint);

 context.ProgressHandler = new Progress<TransferStatus>((progress) =>
 {
 Console.WriteLine("\rBytes transferred: {0}", progress.BytesTransferred);
 });

 return context;
}

public static DirectoryTransferContext GetDirectoryTransferContext(TransferCheckpoint checkpoint)
{
 DirectoryTransferContext context = new DirectoryTransferContext(checkpoint);

 context.ProgressHandler = new Progress<TransferStatus>((progress) =>
 {
 Console.WriteLine("\rBytes transferred: {0}", progress.BytesTransferred);
 });

 return context;
}

```

Modify the `TransferLocalFileToAzureBlob` method to use `GetSingleTransferContext`:

```

public static async Task TransferLocalFileToAzureBlob(CloudStorageAccount account)
{
 string localFilePath = GetSourcePath();
 CloudBlockBlob blob = GetBlob(account);
 TransferCheckpoint checkpoint = null;
 SingleTransferContext context = GetSingleTransferContext(checkpoint);
 Console.WriteLine("\nTransfer started...\n");
 Stopwatch stopWatch = Stopwatch.StartNew();
 await TransferManager.UploadAsync(localFilePath, blob, null, context);
 stopWatch.Stop();
 Console.WriteLine("\nTransfer operation completed in " + stopWatch.Elapsed.TotalSeconds + " seconds.");
 ExecuteChoice(account);
}

```

## Resume a canceled transfer

Another convenient feature offered by the Data Movement library is the ability to resume a canceled transfer. Let's add some code that allows us to temporarily cancel the transfer by typing `c`, and then resume the transfer 3 seconds later.

Modify `TransferLocalFileToAzureBlob`:

```

public static async Task TransferLocalFileToAzureBlob(CloudStorageAccount account)
{
 string localFilePath = GetSourcePath();
 CloudBlockBlob blob = GetBlob(account);
 TransferCheckpoint checkpoint = null;
 SingleTransferContext context = GetSingleTransferContext(checkpoint);
 CancellationTokenSource cancellationSource = new CancellationTokenSource();
 Console.WriteLine("\nTransfer started...\nPress 'c' to temporarily cancel your transfer...\n");

 Stopwatch stopWatch = Stopwatch.StartNew();
 Task task;
 ConsoleKeyInfo keyinfo;
 try
 {
 task = TransferManager.UploadAsync(localFilePath, blob, null, context, cancellationSource.Token);
 while(!task.IsCompleted)
 {
 if(Console.KeyAvailable)
 {
 keyinfo = Console.ReadKey(true);
 if(keyinfo.Key == ConsoleKey.C)
 {
 cancellationSource.Cancel();
 }
 }
 await task;
 }
 }
 catch(Exception e)
 {
 Console.WriteLine("\nThe transfer is canceled: {0}", e.Message);
 }

 if(cancellationSource.IsCancellationRequested)
 {
 Console.WriteLine("\nTransfer will resume in 3 seconds...");
 Thread.Sleep(3000);
 checkpoint = context.LastCheckpoint;
 context = GetSingleTransferContext(checkpoint);
 Console.WriteLine("\nResuming transfer...\n");
 await TransferManager.UploadAsync(localFilePath, blob, null, context);
 }

 stopWatch.Stop();
 Console.WriteLine("\nTransfer operation completed in " + stopWatch.Elapsed.TotalSeconds + " seconds.");
 ExecuteChoice(account);
}

```

Up until now, our `checkpoint` value has always been set to `null`. Now, if we cancel the transfer, we retrieve the last checkpoint of our transfer, then use this new checkpoint in our transfer context.

## Transfer a local directory to Blob storage

It would be disappointing if the Data Movement library could only transfer one file at a time. Luckily, this is not the case. The Data Movement library provides the ability to transfer a directory of files and all of its subdirectories. Let's add some code that allows us to do just that.

First, add the method `GetBlobDirectory` to `Program.cs`:

```
public static CloudBlobDirectory GetBlobDirectory(CloudStorageAccount account)
{
 CloudBlobClient blobClient = account.CreateCloudBlobClient();

 Console.WriteLine("\nProvide name of Blob container. This can be a new or existing Blob container:");
 string containerName = Console.ReadLine();
 CloudBlobContainer container = blobClient.GetContainerReference(containerName);
 container.CreateIfNotExistsAsync().Wait();

 CloudBlobDirectory blobDirectory = container.GetDirectoryReference("");
 return blobDirectory;
}
```

Then, modify `TransferLocalDirectoryToAzureBlobDirectory` :

```

public static async Task TransferLocalDirectoryToAzureBlobDirectory(CloudStorageAccount account)
{
 string localDirectoryPath = GetSourcePath();
 CloudBlobDirectory blobDirectory = GetBlobDirectory(account);
 TransferCheckpoint checkpoint = null;
 DirectoryTransferContext context = GetDirectoryTransferContext(checkpoint);
 CancellationTokenSource cancellationSource = new CancellationTokenSource();
 Console.WriteLine("\nTransfer started...\\nPress 'c' to temporarily cancel your transfer...\\n");

 Stopwatch stopWatch = Stopwatch.StartNew();
 Task task;
 ConsoleKeyInfo keyinfo;
 UploadDirectoryOptions options = new UploadDirectoryOptions()
 {
 Recursive = true
 };

 try
 {
 task = TransferManager.UploadDirectoryAsync(localDirectoryPath, blobDirectory, options, context,
cancellationSource.Token);
 while(!task.IsCompleted)
 {
 if(Console.KeyAvailable)
 {
 keyinfo = Console.ReadKey(true);
 if(keyinfo.Key == ConsoleKey.C)
 {
 cancellationSource.Cancel();
 }
 }
 }
 await task;
 }
 catch(Exception e)
 {
 Console.WriteLine("\nThe transfer is canceled: {0}", e.Message);
 }

 if(cancellationSource.IsCancellationRequested)
 {
 Console.WriteLine("\nTransfer will resume in 3 seconds...");
 Thread.Sleep(3000);
 checkpoint = context.LastCheckpoint;
 context = GetDirectoryTransferContext(checkpoint);
 Console.WriteLine("\nResuming transfer...\\n");
 await TransferManager.UploadDirectoryAsync(localDirectoryPath, blobDirectory, options, context);
 }

 stopWatch.Stop();
 Console.WriteLine("\nTransfer operation completed in " + stopWatch.Elapsed.TotalSeconds + " seconds.");
 ExecuteChoice(account);
}

```

There are a few differences between this method and the method for uploading a single file. We're now using `TransferManager.UploadDirectoryAsync` and the `GetDirectoryTransferContext` method we created earlier. In addition, we now provide an `options` value to our upload operation, which allows us to indicate that we want to include subdirectories in our upload.

## Copy a file from URL to a blob

Now, let's add code that allows us to copy a file from a URL to an Azure Blob.

Modify `TransferUrlToAzureBlob`:

```

public static async Task TransferUrlToAzureBlob(CloudStorageAccount account)
{
 Uri uri = new Uri(GetSourcePath());
 CloudBlockBlob blob = GetBlob(account);
 TransferCheckpoint checkpoint = null;
 SingleTransferContext context = GetSingleTransferContext(checkpoint);
 CancellationTokenSource cancellationSource = new CancellationTokenSource();
 Console.WriteLine("\nTransfer started...\nPress 'c' to temporarily cancel your transfer...\n");

 Stopwatch stopWatch = Stopwatch.StartNew();
 Task task;
 ConsoleKeyInfo keyinfo;
 try
 {
 task = TransferManager.CopyAsync(uri, blob, true, null, context, cancellationSource.Token);
 while(!task.IsCompleted)
 {
 if(Console.KeyAvailable)
 {
 keyinfo = Console.ReadKey(true);
 if(keyinfo.Key == ConsoleKey.C)
 {
 cancellationSource.Cancel();
 }
 }
 await task;
 }
 }
 catch(Exception e)
 {
 Console.WriteLine("\nThe transfer is canceled: {0}", e.Message);
 }

 if(cancellationSource.IsCancellationRequested)
 {
 Console.WriteLine("\nTransfer will resume in 3 seconds...");
 Thread.Sleep(3000);
 checkpoint = context.LastCheckpoint;
 context = GetSingleTransferContext(checkpoint);
 Console.WriteLine("\nResuming transfer...\n");
 await TransferManager.CopyAsync(uri, blob, true, null, context, cancellationSource.Token);
 }

 stopWatch.Stop();
 Console.WriteLine("\nTransfer operation completed in " + stopWatch.Elapsed.TotalSeconds + " seconds.");
 ExecuteChoice(account);
}

```

One important use case for this feature is when you need to move data from another cloud service (e.g. AWS) to Azure. As long as you have a URL that gives you access to the resource, you can easily move that resource into Azure Blobs by using the `TransferManager.CopyAsync` method. This method also introduces a new boolean parameter. Setting this parameter to `true` indicates that we want to do an asynchronous server-side copy. Setting this parameter to `false` indicates a synchronous copy - meaning the resource is downloaded to our local machine first, then uploaded to Azure Blob. However, synchronous copy is currently only available for copying from one Azure Storage resource to another.

## Copy a blob

Another feature that's uniquely provided by the Data Movement library is the ability to copy from one Azure Storage resource to another.

Modify `TransferAzureBlobToAzureBlob`:

```

public static async Task TransferAzureBlobToAzureBlob(CloudStorageAccount account)
{
 CloudBlockBlob sourceBlob = GetBlob(account);
 CloudBlockBlob destinationBlob = GetBlob(account);
 TransferCheckpoint checkpoint = null;
 SingleTransferContext context = GetSingleTransferContext(checkpoint);
 CancellationTokenSource cancellationSource = new CancellationTokenSource();
 Console.WriteLine("\nTransfer started...\nPress 'c' to temporarily cancel your transfer...\n");

 Stopwatch stopWatch = Stopwatch.StartNew();
 Task task;
 ConsoleKeyInfo keyinfo;
 try
 {
 task = TransferManager.CopyAsync(sourceBlob, destinationBlob, CopyMethod.ServiceSideAsyncCopy, null,
context, cancellationSource.Token);
 while(!task.IsCompleted)
 {
 if(Console.KeyAvailable)
 {
 keyinfo = Console.ReadKey(true);
 if(keyinfo.Key == ConsoleKey.C)
 {
 cancellationSource.Cancel();
 }
 }
 }
 await task;
 }
 catch(Exception e)
 {
 Console.WriteLine("\nThe transfer is canceled: {0}", e.Message);
 }

 if(cancellationSource.IsCancellationRequested)
 {
 Console.WriteLine("\nTransfer will resume in 3 seconds...");
 Thread.Sleep(3000);
 checkpoint = context.LastCheckpoint;
 context = GetSingleTransferContext(checkpoint);
 Console.WriteLine("\nResuming transfer...\n");
 await TransferManager.CopyAsync(sourceBlob, destinationBlob, false, null, context,
cancellationSource.Token);
 }

 stopWatch.Stop();
 Console.WriteLine("\nTransfer operation completed in " + stopWatch.Elapsed.TotalSeconds + " seconds.");
 ExecuteChoice(account);
}

```

In this example, we set the boolean parameter in `TransferManager.CopyAsync` to `false` to indicate that we want to do a synchronous copy. This means that the resource is downloaded to our local machine first, then uploaded to Azure Blob. The synchronous copy option is a great way to ensure that your copy operation has a consistent speed. In contrast, the speed of an asynchronous server-side copy is dependent on the available network bandwidth on the server, which can fluctuate. However, synchronous copy may generate additional egress cost compared to asynchronous copy. The recommended approach is to use synchronous copy in an Azure VM that is in the same region as your source storage account to avoid egress cost.

The data movement application is now complete. [The full code sample is available on GitHub.](#)

## Next steps

[Azure Storage Data Movement library reference documentation.](#)

**TIP**

Manage Azure Blob storage resources with Azure Storage Explorer. [Azure Storage Explorer](#) is a free, standalone app from Microsoft that enables you to [manage Azure Blob storage resources](#). Using Azure Storage Explorer, you can visually create, read, update, and delete blob containers and blobs, as well as manage access to your blobs containers and blobs.

# Use the Azure Identity library to get an access token for authorization

8/22/2022 • 6 minutes to read • [Edit Online](#)

The Azure Identity client library simplifies the process of getting an OAuth 2.0 access token for authorization with Azure Active Directory (Azure AD) via the [Azure SDK](#). The latest versions of the Azure Storage client libraries for .NET, Java, Python, JavaScript, and Go integrate with the Azure Identity libraries for each of those languages to provide a simple and secure means to acquire an access token for authorization of Azure Storage requests.

An advantage of the Azure Identity client library is that it enables you to use the same code to acquire the access token whether your application is running in the development environment or in Azure. The Azure Identity client library returns an access token for a security principal. When your code is running in Azure, the security principal may be a managed identity for Azure resources, a service principal, or a user or group. In the development environment, the client library provides an access token for either a user or a service principal for testing purposes.

The access token returned by the Azure Identity client library is encapsulated in a token credential. You can then use the token credential to get a service client object to use in performing authorized operations against Azure Storage. A simple way to get the access token and token credential is to use the `DefaultAzureCredential` class that is provided by the Azure Identity client library. An instance of this class attempts to get the token credential in a variety of common ways, and it works in both the development environment and in Azure.

For more information about the Azure Identity client library for your language, see one of the following articles:

- [Azure Identity client library for .NET](#)
- [Azure Identity client library for Java](#)
- [Azure Identity client library for Python](#)
- [Azure Identity client library for JavaScript](#)
- [Azure Identity client library for Go](#)

## Assign Azure roles for access to data

When an Azure AD security principal attempts to access data in an Azure Storage account, that security principal must have permissions to the data resource. Whether the security principal is a managed identity in Azure or an Azure AD user account running code in the development environment, the security principal must be assigned an Azure role that grants access to data in Azure Storage. For information about assigning permissions for data access via Azure RBAC, see one of the following articles:

- [Assign an Azure role for access to blob data](#)
- [Assign an Azure role for access to queue data](#)
- [Assign an Azure role for access to table data](#)

### NOTE

When you create an Azure Storage account, you are not automatically assigned permissions to access data via Azure AD. You must explicitly assign yourself an Azure role for Azure Storage. You can assign it at the level of your subscription, resource group, storage account, or container, queue, or table.

# Authenticate the user in the development environment

When your code is running in the development environment, authentication may be handled automatically, or it may require a browser login, depending on which tools you're using. For example, Microsoft Visual Studio and Microsoft Visual Studio Code support single sign-on (SSO), so that the active Azure AD user account is automatically used for authentication. For more information about SSO, see [Single sign-on to applications](#).

You can also create a service principal and set environment variables that the development environment can read at runtime.

## Authenticate a service principal in the development environment

If your development environment does not support single sign-on or login via a web browser, then you can use a service principal to authenticate from the development environment. After you create the service principal, add the values returned for the service principal to environment variables.

### Create the service principal

To create a service principal with Azure CLI and assign an Azure role, call the `az ad sp create-for-rbac` command. Provide an Azure Storage data access role to assign to the new service principal. Additionally, provide the scope for the role assignment. For more information about the built-in roles provided for Azure Storage, see [Azure built-in roles](#).

If you do not have sufficient permissions to assign a role to the service principal, you may need to ask the account owner or administrator to perform the role assignment.

The following example uses the Azure CLI to create a new service principal and assign the **Storage Blob Data Contributor** role to it, scoped to the storage account:

```
az ad sp create-for-rbac \
 --name <service-principal> \
 --role "Storage Blob Data Contributor" \
 --scopes /subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>
```

The `az ad sp create-for-rbac` command returns a list of service principal properties in JSON format. Copy these values so that you can use them to create the necessary environment variables in the next step.

```
{
 "appId": "generated-app-ID",
 "displayName": "service-principal-name",
 "name": "http://service-principal-uri",
 "password": "generated-password",
 "tenant": "tenant-ID"
}
```

For more information about how to create a service principal, see one of the following articles:

- [Create an Azure AD app and service principal in the portal](#)
- [Create an Azure service principal with Azure PowerShell](#)
- [Create an Azure service principal with Azure CLI](#)

### IMPORTANT

Azure role assignments may take several minutes to propagate.

## Set environment variables

The Azure Identity client library reads values from three environment variables at runtime to authenticate the service principal. The following table describes the value to set for each environment variable.

ENVIRONMENT VARIABLE	VALUE
AZURE_CLIENT_ID	The app ID for the service principal
AZURE_TENANT_ID	The service principal's Azure AD tenant ID
AZURE_CLIENT_SECRET	The password generated for the service principal

### IMPORTANT

After you set the environment variables, close and re-open your console window. If you are using Visual Studio or another development environment, you may need to restart the development environment in order for it to register the new environment variables.

For more information, see [Create identity for Azure app in portal](#).

## Get an access token for authorization

The Azure Identity client library provides classes that you can use to get an access token for authorizing requests with Azure AD. The Azure Storage client libraries include constructors for service client objects that take a token credential as a parameter. You can use the two together to easily authorize Azure Storage requests with Azure AD credentials.

Using the **DefaultAzureCredential** class is recommended for most simple scenarios where your application needs to get an access token in both the development environment and in Azure. A variety of types of token credentials are also available for other scenarios.

The following example shows how to use DefaultAzureCredential to get a token in .NET. The application then uses the token to create a new service client, which is then used to create a blob container. Although this example uses .NET and the Blob Storage service, the **DefaultAzureCredential** class behaves similarly with other languages and with other Azure services.

```
static void CreateBlobContainer(string accountName, string containerName)
{
 // Construct the blob container endpoint from the arguments.
 string containerEndpoint = string.Format("https://'{0}'.blob.core.windows.net/{1}",
 accountName,
 containerName);

 // Get a token credential and create a service client object for the blob container.
 BlobContainerClient containerClient = new BlobContainerClient(new Uri(containerEndpoint),
 new DefaultAzureCredential());

 // Create the container if it does not exist.
 containerClient.CreateIfNotExists();
}
```

For more information about using the **DefaultAzureCredential** class to authorize a managed identity to access Azure Storage, see [Azure Identity client library for .NET](#).

## See also

- Apps & service principals in Azure AD
- Microsoft identity platform authentication libraries

# Authorize access to blob data with managed identities for Azure resources

8/22/2022 • 3 minutes to read • [Edit Online](#)

Azure Blob Storage supports Azure Active Directory (Azure AD) authentication with [managed identities for Azure resources](#). Managed identities for Azure resources can authorize access to blob data using Azure AD credentials from applications running in Azure virtual machines (VMs), function apps, virtual machine scale sets, and other services. By using managed identities for Azure resources together with Azure AD authentication, you can avoid storing credentials with your applications that run in the cloud.

This article shows how to authorize access to blob data from an Azure VM using managed identities for Azure Resources.

## Enable managed identities on a VM

Before you can use managed identities for Azure Resources to authorize access to blobs from your VM, you must first enable managed identities for Azure Resources on the VM. To learn how to enable managed identities for Azure Resources, see one of these articles:

- [Azure portal](#)
- [Azure PowerShell](#)
- [Azure CLI](#)
- [Azure Resource Manager template](#)
- [Azure Resource Manager client libraries](#)

For more information about managed identities, see [Managed identities for Azure resources](#).

## Assign an RBAC role to a managed identity

When an Azure AD security principal attempts to access data in an Azure Storage account, that security principal must have permissions to the data resource. Whether the security principal is a managed identity in Azure or an Azure AD user account running code in the development environment, the security principal must be assigned an Azure role that grants access to data in Azure Storage. For information about assigning permissions for data access via Azure RBAC, see [Assign an Azure role for access to blob data](#).

### NOTE

When you create an Azure Storage account, you are not automatically assigned permissions to access data via Azure AD. You must explicitly assign yourself an Azure role for Azure Storage. You can assign it at the level of your subscription, resource group, storage account, or container.

## Use a managed identity to create a block blob in .NET

The Azure Identity client library simplifies the process of getting an OAuth 2.0 access token for authorization with Azure Active Directory (Azure AD) via the [Azure SDK](#). When you use the Azure Identity client library to get an access token, you can use the same code to acquire the token whether your application is running in the development environment or in Azure. For more information, see [Use the Azure Identity library to get an access token for authorization](#).

To get a token that your code can use to authorize requests to Azure Storage, create an instance of the [DefaultAzureCredential](#) class. You can then use the token to create a service client object that is authorized to perform data operations in Azure Storage. For more information about using the [DefaultAzureCredential](#) class to authorize a managed identity to access Azure Storage, see [Azure Identity client library for .NET](#).

The following code example shows how to get an access token and use it to create a service client object, then uses the service client to upload a new blob:

```
async static Task CreateBlockBlobAsync(string accountName, string containerName, string blobName)
{
 // Construct the blob container endpoint from the arguments.
 string containerEndpoint = string.Format("https://{}{}.blob.core.windows.net/{}", accountName, containerName);

 // Get a credential and create a service client object for the blob container.
 BlobContainerClient containerClient = new BlobContainerClient(new Uri(containerEndpoint),
 new DefaultAzureCredential());

 try
 {
 // Create the container if it does not exist.
 await containerClient.CreateIfNotExistsAsync();

 // Upload text to a new block blob.
 string blobContents = "This is a block blob.";
 byte[] byteArray = Encoding.ASCII.GetBytes(blobContents);

 using (MemoryStream stream = new MemoryStream(byteArray))
 {
 await containerClient.UploadBlobAsync(blobName, stream);
 }
 }
 catch (RequestFailedException e)
 {
 Console.WriteLine(e.Message);
 Console.ReadLine();
 throw;
 }
}
```

#### NOTE

To authorize requests against blob data with Azure AD, you must use HTTPS for those requests.

## Next steps

- [Assign an Azure role for access to blob data](#)
- [Authorize access to blob or queue data from a native or web application](#)
- [Tutorial: Access storage from App Service using managed identities](#)

# Authorize access to blob or queue data from a native or web application

8/22/2022 • 12 minutes to read • [Edit Online](#)

A key advantage of using Azure Active Directory (Azure AD) with Azure Blob storage or Queue storage is that your credentials no longer need to be stored in your code. Instead, you can request an OAuth 2.0 access token from the Microsoft identity platform. Azure AD authenticates the security principal (a user, group, or service principal) running the application. If authentication succeeds, Azure AD returns the access token to the application, and the application can then use the access token to authorize requests to Azure Blob storage or Queue storage.

This article shows how to configure your native application or web application for authentication with the Microsoft identity platform using a sample application that is available for download. The sample application features .NET, but other languages use a similar approach. For more information about the Microsoft identity platform, see [Microsoft identity platform overview](#).

For an overview of the OAuth 2.0 code grant flow, see [Authorize access to Azure Active Directory web applications using the OAuth 2.0 code grant flow](#).

## About the sample application

The sample application provides an end-to-end experience that shows how to configure a web application for authentication with Azure AD in a local development environment. To view and run the sample application, first clone or download it from [GitHub](#). Then follow the steps outlined in the article to configure an Azure app registration and update the application for your environment.

## Assign a role to an Azure AD security principal

To authenticate a security principal from your Azure Storage application, first configure Azure role-based access control (Azure RBAC) settings for that security principal. Azure Storage defines built-in roles that encompass permissions for containers and queues. When the Azure role is assigned to a security principal, that security principal is granted access to that resource. For more information, see [Assign an Azure role for access to blob data](#).

## Register your application with an Azure AD tenant

The first step in using Azure AD to authorize access to storage resources is registering your client application with an Azure AD tenant from the [Azure portal](#). When you register your client application, you supply information about the application to Azure AD. Azure AD then provides a client ID (also called an *application ID*) that you use to associate your application with Azure AD at runtime. To learn more about the client ID, see [Application and service principal objects in Azure Active Directory](#). To register your Azure Storage application, follow the steps shown in [Quickstart: Register an application with the Microsoft identity platform](#).

The following image shows common settings for registering a web application. Note that in this example, the redirect URI is set to `http://localhost:5000/signin-oidc` for testing the sample application in the development environment. You can modify this setting later under the **Authentication** setting for your registered application in the Azure portal:

## Register an application

X

\* Name

The user-facing display name for this application (this can be changed later).

✓

Supported account types

Who can use this application or access this API?

- Accounts in this organizational directory only (Default Directory only - Single tenant)
- Accounts in any organizational directory (Any Azure AD directory - Multitenant)
- Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)
- Personal Microsoft accounts only

[Help me choose...](#)

Redirect URI (optional)

We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Web	<input type="text" value="http://localhost:5000/signin-oidc"/>	✓
-----	----------------------------------------------------------------	---

[By proceeding, you agree to the Microsoft Platform Policies](#) ↗

**Register**

**NOTE**

If you register your application as a native application, you can specify any valid URI for the **Redirect URI**. For native applications, this value does not have to be a real URL. For web applications, the redirect URI must be a valid URI, because it specifies the URL to which tokens are provided.

After you've registered your application, you'll see the application ID (or client ID) under **Settings**:

Dashboard > Microsoft - App registrations > StorageClientAppSample

Overview	<div style="display: flex; justify-content: space-between;"> <div style="flex: 1;"> <div style="display: flex; align-items: center;"> <span style="font-size: 2em; margin-right: 10px;">StorageClientAppSample</span> <span style="margin-left: auto;">↗</span> </div> <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div style="flex: 1;"> <input type="text" value="Search (Ctrl+I)"/> </div> <div style="flex: 1; text-align: right;"> <span style="color: #0078d4;">Delete</span> <span style="color: #0078d4;">Endpoints</span> </div> </div> </div> <div style="display: flex; justify-content: space-between; margin-top: 20px;"> <div style="flex: 1;"> <p>Display name StorageClientAppSample</p> <p>Application (client) ID &lt;application-id&gt;</p> <p>Directory (tenant) ID &lt;directory-id&gt;</p> <p>Object ID &lt;object-id&gt;</p> </div> <div style="flex: 1;"> <p>Supported account types My organization only</p> <p>Redirect URIs 1 web, 0 public client</p> <p>Managed application in local directory <a href="#">Create Service Principal</a></p> </div> </div> </div>
<ul style="list-style-type: none"> <li><span style="color: #0078d4;">Overview</span></li> <li><span style="color: #0078d4;">Quickstart</span></li> <li><span style="color: #0078d4;">Manage</span></li> <li><span style="color: #0078d4;">Branding</span></li> <li><span style="color: #0078d4;">Authentication</span></li> </ul>	

For more information about registering an application with Azure AD, see [Integrating applications with Azure Active Directory](#).

### Grant your registered app permissions to Azure Storage

Next, grant your application permissions to call Azure Storage APIs. This step enables your application to authorize requests to Azure Storage with Azure AD.

1. On the API permissions page for your registered application, select **Add a permission**.
2. Under the Microsoft APIs tab, select **Azure Storage**.
3. On Request API permissions pane, under **What type of permissions does your application require?**, observe that the available permission type is **Delegated permissions**. This option is selected for you by default.
4. Under **Permissions**, select the checkbox next to **user\_impersonation**, then select the **Add permissions** button.

**Request API permissions**

[All APIs](#) [Azure Storage](#) <https://storage.azure.com/> [Docs](#)

What type of permissions does your application require?

**Delegated permissions**  
Your application needs to access the API as the signed-in user.

**Application permissions**  
Your application runs as a background service or daemon without a signed-in user.

Select permissions [expand all](#)

Start typing a reply url to filter these results

Permission	Admin consent required
<input checked="" type="checkbox"/> <b>user_impersonation</b> ⓘ Access Azure Storage	-

**Add permissions** **Discard**

5. Next, grant admin consent for these permissions by clicking **Grant admin consent for Default Directory**.

The API permissions pane now shows that your registered Azure AD application has access to both the Microsoft Graph and Azure Storage APIs, and that consent is granted for the default directory. Permissions are granted to Microsoft Graph automatically when you first register your app with Azure AD.

Home > App registrations > StorageClientAppSample

**StorageClientAppSample | API permissions**

Search (Ctrl+ /) [Refresh](#) [Got feedback?](#)

Successfully granted admin consent for the requested permissions.

Configured permissions

Applications are authorized to call APIs when they are granted permissions by users/admins as part of the consent process. The list of configured permissions should include all the permissions the application needs. [Learn more about permissions and consent](#)

+ Add a permission	<input checked="" type="checkbox"/> Grant admin consent for Default Directory			
<b>API / Permissions name</b> Type Description Admin consent req... Status				
Azure Storage (1)				
user_impersonation	Delegated	Access Azure Storage	-	<input checked="" type="checkbox"/> Granted for Default Dire...
Microsoft Graph (1)				
User.Read	Delegated	Sign in and read user profile	-	<input checked="" type="checkbox"/> Granted for Default Dire...

## Create a client secret

The application needs a client secret to prove its identity when requesting a token. For security reasons, Microsoft limits creation of client secrets longer than 24 months and strongly recommends that you set this to a value less than 12 months. To add the client secret, follow these steps:

1. Navigate to your app registration in the Azure portal.
2. Select the **Certificates & secrets** setting.
3. Under **Client secrets**, click **New client secret** to create a new secret.
4. Provide a description for the secret, and choose the desired expiration interval.
5. Immediately copy the value of the new secret to a secure location. The full value is displayed to you only once.

**Client secrets**

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

DESCRIPTION	EXPIRES	VALUE	
storage auth flow sample	6/6/2020	+oN*****	

## Enable implicit grant flow

Next, configure implicit grant flow for your application. Follow these steps:

1. Navigate to your app registration in the Azure portal.
2. In the **Manage** section, select the **Authentication** setting.
3. In the **Implicit grant** section, select the check box to enable ID tokens, as shown in the following image:

**Implicit grant**

Allows an application to request a token directly from the authorization endpoint. Checking Access tokens and ID tokens is recommended only if the application has a single-page architecture (SPA), has no back-end components, does not use the latest version of MSAL.js with auth code flow, or it invokes a web API via JavaScript. ID Token is needed for ASP.NET Core Web Apps. [Learn more about the implicit grant flow](#)

To enable the implicit grant flow, select the tokens you would like to be issued by the authorization endpoint:

Access tokens  
 ID tokens

## Client libraries for token acquisition

Once you have registered your application and granted it permissions to access data in Azure Blob storage or Queue storage, you can add code to your application to authenticate a security principal and acquire an OAuth 2.0 token. To authenticate and acquire the token, you can use either one of the [Microsoft identity platform authentication libraries](#) or another open-source library that supports OpenID Connect 1.0. Your application can then use the access token to authorize a request against Azure Blob storage or Queue storage.

For a list of scenarios for which acquiring tokens is supported, see the [authentication flows](#) section of the [Microsoft Authentication Library \(MSAL\)](#) documentation.

## Well-known values for authentication with Azure AD

To authenticate a security principal with Azure AD, you need to include some well-known values in your code.

## Azure AD authority

For Microsoft public cloud, the base Azure AD authority is as follows, where *tenant-id* is your Active Directory tenant ID (or directory ID):

```
https://login.microsoftonline.com/<tenant-id>/
```

The tenant ID identifies the Azure AD tenant to use for authentication. It is also referred to as the directory ID. To retrieve the tenant ID, navigate to the [Overview](#) page for your app registration in the Azure portal, and copy the value from there.

## Azure Storage resource ID

An Azure AD resource ID indicates the audience for which a token that is issued can be used to provide access to an Azure resource. In the case of Azure Storage, the resource ID may be specific to a single storage account, or it may apply to any storage account. The following table describes the values that you can provide for the resource ID:

RESOURCE ID	DESCRIPTION
<code>https://&lt;account&gt;.blob.core.windows.net</code>	The service endpoint for a given storage account. Use this value to acquire a token for authorizing requests to that specific Azure Storage account and service only. Replace the value in brackets with the name of your storage account.
<code>https://&lt;account&gt;.queue.core.windows.net</code>	
<code>https://storage.azure.com/</code>	Use to acquire a token for authorizing requests to any Azure Storage account.

## .NET code example: Create a block blob

The code example shows how to get an access token from Azure AD. The access token is used to authenticate the specified user and then authorize a request to create a block blob. To get this sample working, first follow the steps outlined in the preceding sections.

To request the token, you will need the following values from your app's registration:

- The name of your Azure AD domain. Retrieve this value from the [Overview](#) page of your Azure Active Directory.
- The tenant (or directory) ID. Retrieve this value from the [Overview](#) page of your app registration.
- The client (or application) ID. Retrieve this value from the [Overview](#) page of your app registration.
- The client redirection URI. Retrieve this value from the [Authentication](#) settings for your app registration.
- The value of the client secret. Retrieve this value from the location to which you previously copied it.

### Create a storage account and container

To run the code sample, create a storage account within the same subscription as your Azure Active Directory. Then create a container within that storage account. The sample code will create a block blob in this container.

Next, explicitly assign the **Storage Blob Data Contributor** role to the user account under which you will run the sample code. To learn how to assign this role in the Azure portal, see [Assign an Azure role for access to blob data](#).

#### **NOTE**

When you create an Azure Storage account, you are not automatically assigned permissions to access data via Azure AD. You must explicitly assign yourself an Azure role for Azure Storage. You can assign it at the level of your subscription, resource group, storage account, or container or queue.

Prior to assigning yourself a role for data access, you will be able to access data in your storage account via the Azure portal because the Azure portal can also use the account key for data access. For more information, see [Choose how to authorize access to blob data in the Azure portal](#).

### **Create a web application that authorizes access to Blob storage with Azure AD**

When your application accesses Azure Storage, it does so on the user's behalf, meaning that blob or queue resources are accessed using the permissions of the user who is logged in. To try this code example, you need a web application that prompts the user to sign in using an Azure AD identity. You can create your own, or use the sample application provided by Microsoft.

A completed sample web application that acquires a token and uses it to create a blob in Azure Storage is available on [GitHub](#). Reviewing and running the completed sample may be helpful for understanding the code examples. For instructions about how to run the completed sample, see the section titled [View and run the completed sample](#).

#### **Add references and using statements**

From Visual Studio, install the Azure Storage client library. From the **Tools** menu, select **NuGet Package Manager**, then **Package Manager Console**. Type the following commands into the console window to install the necessary packages from the Azure Storage client library for .NET:

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

```
Install-Package Azure.Storage.Blobs
Install-Package Microsoft.Identity.Web -Version 0.4.0-preview
```

Next, add the following using statements to the HomeController.cs file:

```
using Microsoft.Identity.Web; //MSAL library for getting the access token
using Azure.Storage.Blobs;
```

#### **Create a block blob**

Add the following code snippet to create a block blob. Remember to replace values in angle brackets with your own values:

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

```

private static async Task<string> CreateBlob(TokenAcquisitionTokenCredential tokenCredential)
{
 Uri blobUri = new Uri("https://<storage-account>.blob.core.windows.net/<container>/Blob1.txt");
 BlobClient blobClient = new BlobClient(blobUri, tokenCredential);

 string blobContents = "Blob created by Azure AD authenticated user.";
 byte[] byteArray = Encoding.ASCII.GetBytes(blobContents);

 using (MemoryStream stream = new MemoryStream(byteArray))
 {
 await blobClient.UploadAsync(stream);
 }
 return "Blob successfully created";
}

```

#### NOTE

To authorize blob and queue operations with an OAuth 2.0 token, you must use HTTPS.

In the example above, the .NET client library handles the authorization of the request to create the block blob. Azure Storage client libraries for other languages also handle the authorization of the request for you. However, if you are calling an Azure Storage operation with an OAuth token using the REST API, then you'll need to construct the **Authorization** header by using the OAuth token.

To call Blob and Queue service operations using OAuth access tokens, pass the access token in the **Authorization** header using the **Bearer** scheme, and specify a service version of 2017-11-09 or higher, as shown in the following example:

```

GET /container/file.txt HTTP/1.1
Host: mystorageaccount.blob.core.windows.net
x-ms-version: 2017-11-09
Authorization: Bearer eyJ0eXAiOnJKV1...Xd6j

```

#### Get an access token from Azure AD

Next, add a method that requests a token from Azure AD on the behalf of the user. This method defines the scope for which permissions are to be granted. For more information about permissions and scopes, see [Permissions and consent in the Microsoft identity platform endpoint](#).

Use the resource ID to construct the scope for which to acquire the token. The example constructs the scope by using the resource ID together with the built-in `user_impersonation` scope, which indicates that the token is being requested on behalf of the user.

Keep in mind that you may need to present the user with an interface that enables the user to consent to request the token their behalf:

```

[AuthorizeForScopes(Scopes = new string[] { "https://storage.azure.com/user_impersonation" })]
public async Task<IActionResult> Blob()
{
 string message = await CreateBlob(new TokenAcquisitionTokenCredential(_tokenAcquisition));
 ViewData["Message"] = message;
 return View();
}

```

Consent is the process of a user granting authorization to an application to access protected resources on their behalf. The Microsoft identity platform supports incremental consent, meaning that a security principal can request a minimum set of permissions initially and add permissions over time as needed. When your code

requests an access token, specify the scope of permissions that your app needs. For more information about incremental consent, see [Incremental and dynamic consent](#).

## View and run the completed sample

To run the sample application, first clone or download it from [GitHub](#). Then update the application as described in the following sections.

### Provide values in the settings file

Update the *appsettings.json* file with your own values, as follows:

```
{
 "AzureAd": {
 "Instance": "https://login.microsoftonline.com/",
 "Domain": "<azure-ad-domain-name>.onmicrosoft.com",
 "TenantId": "<tenant-id>",
 "ClientId": "<client-id>",
 "ClientSecret": "<client-secret>",
 "ClientCertificates": [
],
 "CallbackPath": "/signin-oidc"
 },
 "Logging": {
 "LogLevel": {
 "Default": "Information",
 "Microsoft": "Warning",
 "Microsoft.Hosting.Lifetime": "Information"
 }
 },
 "AllowedHosts": "*"
}
```

### Update the storage account and container name

In the *HomeController.cs* file, update the URI that references the block blob to use the name of your storage account and container, replacing the values in angle brackets with your own values:

```
https://<storage-account>.blob.core.windows.net/<container>/Blob1.txt
```

## Next steps

- [Microsoft identity platform](#)
- [Assign an Azure role for access to blob data](#)
- [Authorize access to blob data with managed identities for Azure resources](#)

# Get started with Azure Blob Storage and .NET

8/22/2022 • 6 minutes to read • [Edit Online](#)

This article shows you how to connect to Azure Blob Storage by using the Azure Blob Storage client library v12 for .NET. Once connected, your code can operate on containers, blobs, and features of the Blob Storage service.

[Package \(NuGet\)](#) | [Samples](#) | [API reference](#) | [Library source code](#) | [Give Feedback](#)

## Prerequisites

- Azure subscription - [create one for free](#)
- Azure storage account - [create a storage account](#)
- Current [.NET Core SDK](#) for your operating system. Be sure to get the SDK and not the runtime.

## Set up your project

Open a command prompt and change directory (`cd`) into your project folder. Then, install the Azure Blob Storage client library for .NET package by using the `dotnet add package` command.

```
cd myProject
dotnet add package Azure.Storage.Blobs
```

Add these `using` statements to the top of your code file.

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

- [Azure.Storage.Blobs](#): Contains the primary classes (*client objects*) that you can use to operate on the service, containers, and blobs.
- [Azure.Storage.Blobs.Specialized](#): Contains classes that you can use to perform operations specific to a blob type (For example: append blobs).
- [Azure.Storage.Blobs.Models](#): All other utility classes, structures, and enumeration types.

## Connect to Blob Storage

To connect to Blob Storage, create an instance of the [BlobServiceClient](#) class. This object is your starting point. You can use it to operate on the blob service instance and its containers. You can create a [BlobServiceClient](#) by using an account access key, a shared access signature (SAS), or by using an Azure Active Directory (Azure AD) authorization token.

To learn more about each of these authorization mechanisms, see [Authorize access to data in Azure Storage](#).

### Authorize with an account key

Create a [StorageSharedKeyCredential](#) by using the storage account name and account key. Then use that object to initialize a [BlobServiceClient](#).

```

public static void GetBlobServiceClient(ref BlobServiceClient blobServiceClient,
 string accountName, string accountKey)
{
 Azure.Storage.StorageSharedKeyCredential sharedKeyCredential =
 new StorageSharedKeyCredential(accountName, accountKey);

 string blobUri = "https://" + accountName + ".blob.core.windows.net";

 blobServiceClient = new BlobServiceClient
 (new Uri(blobUri), sharedKeyCredential);
}

```

You can also create a [BlobServiceClient](#) by using a connection string.

```
BlobServiceClient blobServiceClient = new BlobServiceClient(connectionString);
```

For information about how to obtain account keys and best practice guidelines for properly managing and safeguarding your keys, see [Manage storage account access keys](#).

#### **Authorize with a SAS token**

Create a [Uri](#) by using the blob service endpoint and SAS token. Then, create a [BlobServiceClient](#) by using the [Uri](#).

```

public static void GetBlobServiceClientSAS(ref BlobServiceClient blobServiceClient,
 string accountName, string sasToken)
{
 string blobUri = "https://" + accountName + ".blob.core.windows.net";

 blobServiceClient = new BlobServiceClient
 (new Uri($"{blobUri}?{sasToken}"), null);
}

```

To generate and manage SAS tokens, see any of these articles:

- [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#)
- [Create an account SAS with .NET](#)
- [Create a service SAS for a container or blob](#)
- [Create a user delegation SAS for a container, directory, or blob with .NET](#)

#### **Authorize with Azure AD**

To authorize with Azure AD, you'll need to use a security principal. Which type of security principal you need depends on where your application runs. Use this table as a guide.

WHERE THE APPLICATION RUNS	SECURITY PRINCIPAL	GUIDANCE
Local machine (developing and testing)	User identity or service principal	<a href="#">Use the Azure Identity library to get an access token for authorization</a>
Azure	Managed identity	<a href="#">Authorize access to blob data with managed identities for Azure resources</a>
Servers or clients outside of Azure	Service principal	<a href="#">Authorize access to blob or queue data from a native or web application</a>

If you're testing on a local machine, or your application will run in Azure virtual machines (VMs), function apps, virtual machine scale sets, or in other Azure services, obtain an OAuth token by creating a

[DefaultAzureCredential](#) instance. Use that object to create a [BlobServiceClient](#).

```
public static void GetBlobServiceClient(ref BlobServiceClient blobServiceClient, string accountName)
{
 TokenCredential credential = new DefaultAzureCredential();

 string blobUri = "https://" + accountName + ".blob.core.windows.net";

 blobServiceClient = new BlobServiceClient(new Uri(blobUri), credential);
}
```

If you plan to deploy the application to servers and clients that run outside of Azure, you can obtain an OAuth token by using other classes in the [Azure Identity client library for .NET](#) which derive from the [TokenCredential](#) class.

This example creates a [ClientSecretCredential](#) instance by using the client ID, client secret, and tenant ID. You can obtain these values when you create an app registration and service principal.

```
public static void GetBlobServiceClientAzureAD(ref BlobServiceClient blobServiceClient,
 string accountName, string clientID, string clientSecret, string tenantID)
{
 TokenCredential credential = new ClientSecretCredential(
 tenantID, clientID, clientSecret, new TokenCredentialOptions());

 string blobUri = "https://" + accountName + ".blob.core.windows.net";

 blobServiceClient = new BlobServiceClient(new Uri(blobUri), credential);
}
```

#### Connect anonymously

If you explicitly enable anonymous access, then your code can connect to Blob Storage without authorize your request. You can create a new service client object for anonymous access by providing the Blob storage endpoint for the account. However, you must also know the name of a container in that account that's available for anonymous access. To learn how to enable anonymous access, see [Configure anonymous public read access for containers and blobs](#).

```
public static void CreateAnonymousBlobClient()
{
 // Create the client object using the Blob storage endpoint for your account.
 BlobServiceClient blobServiceClient = new BlobServiceClient
 (new Uri(@"https://storagesamples.blob.core.windows.net/"));

 // Get a reference to a container that's available for anonymous access.
 BlobContainerClient container = blobServiceClient.GetBlobContainerClient("sample-container");

 // Read the container's properties.
 // Note this is only possible when the container supports full public read access.
 Console.WriteLine(container.GetProperties().Value.LastModified);
 Console.WriteLine(container.GetProperties().Value.ETag);
}
```

Alternatively, if you have the URL to a container that is anonymously available, you can use it to reference the container directly.

```

public static void ListBlobsAnonymously()
{
 // Get a reference to a container that's available for anonymous access.
 BlobContainerClient container = new BlobContainerClient(
 (new Uri(@"https://storagesamples.blob.core.windows.net/sample-container")));

 // List blobs in the container.
 // Note this is only possible when the container supports full public read access.
 foreach (BlobItem blobItem in container.GetBlobs())
 {
 Console.WriteLine(container.GetBlockBlobClient(blobItem.Name).Uri);
 }
}

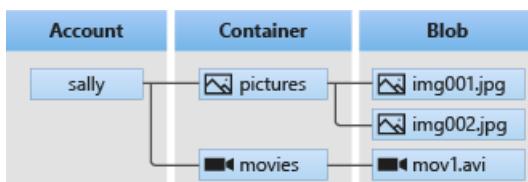
```

## Build your application

As you build your application, your code will primarily interact with three types of resources:

- The storage account, which is the unique top-level namespace for your Azure Storage data.
- Containers, which organize the blob data in your storage account.
- Blobs, which store unstructured data like text and binary data.

The following diagram shows the relationship between these resources.



Each type of resource is represented by one or more associated .NET classes. These are the basic classes:

CLASS	DESCRIPTION
<a href="#">BlobServiceClient</a>	Represents the Blob Storage endpoint for your storage account.
<a href="#">BlobContainerClient</a>	Allows you to manipulate Azure Storage containers and their blobs.
<a href="#">BlobClient</a>	Allows you to manipulate Azure Storage blobs.
<a href="#">AppendBlobClient</a>	Allows you to perform operations specific to append blobs such as periodically appending log data.
<a href="#">BlockBlobClient</a>	Allows you to perform operations specific to block blobs such as staging and then committing blocks of data.

The following guides show you how to use each of these classes to build your application.

GUIDE	DESCRIPTION
<a href="#">Create a container</a>	Create containers.
<a href="#">Delete and restore containers</a>	Delete containers, and if soft-delete is enabled, restore deleted containers.

GUIDE	DESCRIPTION
<a href="#">List containers</a>	List containers in an account and the various options available to customize a listing.
<a href="#">Manage properties and metadata</a>	Get and set properties and metadata for containers.
<a href="#">Create and manage leases</a>	Establish and manage a lock on a container or the blobs in a container.
<a href="#">Append data to blobs</a>	Learn how to create an append blob and then append data to that blob.
<a href="#">Upload blobs</a>	Learn how to upload blobs by using strings, streams, file paths, and other methods.
<a href="#">Download blobs</a>	Download blobs by using strings, streams, and file paths.
<a href="#">Copy blobs</a>	Copy a blob from one account to another account.
<a href="#">List blobs</a>	List blobs in different ways.
<a href="#">Delete and restore</a>	Delete blobs, and if soft-delete is enabled, restore deleted blobs.
<a href="#">Find blobs using tags</a>	Set and retrieve tags as well as use tags to find blobs.
<a href="#">Manage properties and metadata</a>	Get and set properties and metadata for blobs.

## See also

- [Package \(NuGet\)](#)
- [Samples](#)
- [API reference](#)
- [Library source code](#)
- [Give Feedback](#)

# Create a container in Azure Storage with .NET

8/22/2022 • 2 minutes to read • [Edit Online](#)

Blobs in Azure Storage are organized into containers. Before you can upload a blob, you must first create a container. This article shows how to create containers with the [Azure Storage client library for .NET](#).

## Name a container

A container name must be a valid DNS name, as it forms part of the unique URI used to address the container or its blobs. Follow these rules when naming a container:

- Container names can be between 3 and 63 characters long.
- Container names must start with a letter or number, and can contain only lowercase letters, numbers, and the dash (-) character.
- Two or more consecutive dash characters aren't permitted in container names.

The URI for a container is in this format:

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

## Create a container

To create a container, call one of the following methods:

- [CreateBlobContainer](#)
- [CreateBlobContainerAsync](#)

These methods throw an exception if a container with the same name already exists.

Containers are created immediately beneath the storage account. It's not possible to nest one container beneath another.

The following example creates a container asynchronously:

```
//-----
// Create a container
//-----
private static async Task<BlobContainerClient> CreateSampleContainerAsync(BlobServiceClient
blobServiceClient)
{
 // Name the sample container based on new GUID to ensure uniqueness.
 // The container name must be lowercase.
 string containerName = "container-" + Guid.NewGuid();

 try
 {
 // Create the container
 BlobContainerClient container = await blobServiceClient.CreateBlobContainerAsync(containerName);

 if (await container.ExistsAsync())
 {
 Console.WriteLine("Created container {0}", container.Name);
 return container;
 }
 }
 catch (RequestFailedException e)
 {
 Console.WriteLine("HTTP error code {0}: {1}",
 e.Status, e.ErrorCode);
 Console.WriteLine(e.Message);
 }

 return null;
}
```

## Create the root container

A root container serves as a default container for your storage account. Each storage account may have one root container, which must be named `$root`. The root container must be explicitly created or deleted.

You can reference a blob stored in the root container without including the root container name. The root container enables you to reference a blob at the top level of the storage account hierarchy. For example, you can reference a blob that is in the root container in the following manner:

```
https://myaccount.blob.core.windows.net/default.html
```

The following example creates the root container synchronously:

```
//-----
// Create root container
//-----
private static void CreateRootContainer(BlobServiceClient blobServiceClient)
{
 try
 {
 // Create the root container or handle the exception if it already exists
 BlobContainerClient container = blobServiceClient.CreateBlobContainer("$root");

 if (container.Exists())
 {
 Console.WriteLine("Created root container.");
 }
 }
 catch (RequestFailedException e)
 {
 Console.WriteLine("HTTP error code {0}: {1}",
 e.Status, e.ErrorCode);
 Console.WriteLine(e.Message);
 }
}
```

## See also

- [Get started with Azure Blob Storage and .NET](#)
- [Create Container operation](#)
- [Delete Container operation](#)

# Delete and restore a container in Azure Storage with .NET

8/22/2022 • 2 minutes to read • [Edit Online](#)

This article shows how to delete containers with the [Azure Storage client library for .NET](#). If you've enabled container soft delete, you can restore deleted containers.

## Delete a container

To delete a container in .NET, use one of the following methods:

- [Delete](#)
- [DeleteAsync](#)
- [DeleteIfExists](#)
- [DeleteIfExistsAsync](#)

The **Delete** and **DeleteAsync** methods throw an exception if the container doesn't exist.

The **DeleteIfExists** and **DeleteIfExistsAsync** methods return a Boolean value indicating whether the container was deleted. If the specified container doesn't exist, then these methods return **False** to indicate that the container wasn't deleted.

After you delete a container, you can't create a container with the same name for at *least* 30 seconds. Attempting to create a container with the same name will fail with HTTP error code 409 (Conflict). Any other operations on the container or the blobs it contains will fail with HTTP error code 404 (Not Found).

The following example deletes the specified container, and handles the exception if the container doesn't exist:

```

// Delete a container

private static async Task DeleteSampleContainerAsync(BlobServiceClient blobServiceClient, string containerName)
{
 BlobContainerClient container = blobServiceClient.GetBlobContainerClient(containerName);

 try
 {
 // Delete the specified container and handle the exception.
 await container.DeleteAsync();
 }
 catch (RequestFailedException e)
 {
 Console.WriteLine("HTTP error code {0}: {1}",
 e.Status, e.ErrorCode);
 Console.WriteLine(e.Message);
 Console.ReadLine();
 }
}
```

The following example shows how to delete all of the containers that start with a specified prefix.

```

//-----
// Delete all containers with the specified prefix
//-----
private static async Task DeleteContainersWithPrefixAsync(BlobServiceClient blobServiceClient, string
prefix)
{
 Console.WriteLine("Delete all containers beginning with the specified prefix");

 try
 {
 foreach (BlobContainerItem container in blobServiceClient.GetBlobContainers())
 {
 if (container.Name.StartsWith(prefix))
 {
 Console.WriteLine("\tContainer:" + container.Name);
 BlobContainerClient containerClient =
blobServiceClient.GetBlobContainerClient(container.Name);
 await containerClient.DeleteAsync();
 }
 }

 Console.WriteLine();
 }
 catch (RequestFailedException e)
 {
 Console.WriteLine(e.Message);
 Console.ReadLine();
 throw;
 }
}

```

## Restore a deleted container

When container soft delete is enabled for a storage account, a container and its contents may be recovered after it has been deleted, within a retention period that you specify. You can restore a soft deleted container by calling either of the following methods of the [BlobServiceClient](#) class.

- [UndeleteBlobContainer](#)
- [UndeleteBlobContainerAsync](#)

The following example finds a deleted container, gets the version ID of that deleted container, and then passes that ID into the [UndeleteBlobContainerAsync](#) method to restore the container.

```
public static async Task RestoreContainer(BlobServiceClient client, string containerName)
{
 await foreach (BlobContainerItem item in client.GetBlobContainersAsync
 (BlobContainerTraits.None, BlobContainerStates.Deleted))
 {
 if (item.Name == containerName && (item.IsDeleted == true))
 {
 try
 {
 await client.UndeleteBlobContainerAsync(containerName, item.VersionId);
 }
 catch (RequestFailedException e)
 {
 Console.WriteLine("HTTP error code {0}: {1}",
 e.Status, e.ErrorCode);
 Console.WriteLine(e.Message);
 }
 }
 }
}
```

## See also

- [Get started with Azure Blob Storage and .NET](#)
- [Soft delete for containers](#)
- [Enable and manage soft delete for containers](#)
- [Restore Container](#)

# List blob containers with .NET

8/22/2022 • 2 minutes to read • [Edit Online](#)

When you list the containers in an Azure Storage account from your code, you can specify a number of options to manage how results are returned from Azure Storage. This article shows how to list containers using the [Azure Storage client library for .NET](#).

## Understand container listing options

To list containers in your storage account, call one of the following methods:

- [GetBlobContainers](#)
- [GetBlobContainersAsync](#)

The overloads for these methods provide additional options for managing how containers are returned by the listing operation. These options are described in the following sections.

### Manage how many results are returned

By default, a listing operation returns up to 5000 results at a time. To return a smaller set of results, provide a nonzero value for the size of the page of results to return.

If your storage account contains more than 5000 containers, or if you have specified a page size such that the listing operation returns a subset of containers in the storage account, then Azure Storage returns a *continuation token* with the list of containers. A continuation token is an opaque value that you can use to retrieve the next set of results from Azure Storage.

In your code, check the value of the continuation token to determine whether it is empty. When the continuation token is empty, then the set of results is complete. If the continuation token is not empty, then call the listing method again, passing in the continuation token to retrieve the next set of results, until the continuation token is empty.

### Filter results with a prefix

To filter the list of containers, specify a string for the `prefix` parameter. The prefix string can include one or more characters. Azure Storage then returns only the containers whose names start with that prefix.

### Return metadata

To return container metadata with the results, specify the `Metadata` value for the `BlobContainerTraits` enum. Azure Storage includes metadata with each container returned, so you do not need to also fetch the container metadata.

## Example: List containers

The following example asynchronously lists the containers in a storage account that begin with a specified prefix. The example lists containers that begin with the specified prefix and returns the specified number of results per call to the listing operation. It then uses the continuation token to get the next segment of results. The example also returns container metadata with the results.

```
async static Task ListContainers(BlobServiceClient blobServiceClient,
 string prefix,
 int? segmentSize)
{
 try
 {
 // Call the listing operation and enumerate the result segment.
 var resultSegment =
 blobServiceClient.GetBlobContainersAsync(BlobContainerTraits.Metadata, prefix, default)
 .AsPages(default, segmentSize);

 await foreach (Azure.Page<BlobContainerItem> containerPage in resultSegment)
 {
 foreach (BlobContainerItem containerItem in containerPage.Values)
 {
 Console.WriteLine("Container name: {0}", containerItem.Name);
 }

 Console.WriteLine();
 }
 }
 catch (RequestFailedException e)
 {
 Console.WriteLine(e.Message);
 Console.ReadLine();
 throw;
 }
}
```

## See also

- [Get started with Azure Blob Storage and .NET](#)
- [List Containers](#)
- [Enumerating Blob Resources](#)

# Manage container properties and metadata with .NET

8/22/2022 • 2 minutes to read • [Edit Online](#)

Blob containers support system properties and user-defined metadata, in addition to the data they contain. This article shows how to manage system properties and user-defined metadata with the [Azure Storage client library for .NET](#).

## 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 do not affect how the resource behaves.

Metadata name/value pairs are valid HTTP headers, and so should adhere to all restrictions governing HTTP headers. Metadata names must be valid HTTP header names and valid C# identifiers, may contain only ASCII characters, and should be treated as case-insensitive. Metadata values containing non-ASCII characters should be Base64-encoded or URL-encoded.

## Retrieve container properties

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

- [GetProperties](#)
- [GetPropertiesAsync](#)

The following code example fetches a container's system properties and writes some property values to a console window:

```
private static async Task ReadContainerPropertiesAsync(BlobContainerClient container)
{
 try
 {
 // Fetch some container properties and write out their values.
 var properties = await container.GetPropertiesAsync();
 Console.WriteLine($"Properties for container {container.Uri}");
 Console.WriteLine($"Public access level: {properties.Value.PublicAccess}");
 Console.WriteLine($"Last modified time in UTC: {properties.Value.LastModified}");
 }
 catch (RequestFailedException e)
 {
 Console.WriteLine($"HTTP error code {e.Status}: {e.ErrorCode}");
 Console.WriteLine(e.Message);
 Console.ReadLine();
 }
}
```

## 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 to write the values:

- [SetMetadata](#)
- [SetMetadataAsync](#)

The name of your metadata must conform to the naming conventions for C# identifiers. Metadata names preserve the case with which they were created, but are case-insensitive when set or read. If two or more metadata headers with the same name are submitted for a resource, Blob storage comma-separates and concatenates the two values and return HTTP response code 200 (OK).

The following code example sets metadata on a container.

```
public static async Task AddContainerMetadataAsync(BlobContainerClient container)
{
 try
 {
 IDictionary<string, string> metadata =
 new Dictionary<string, string>();

 // Add some metadata to the container.
 metadata.Add("docType", "textDocuments");
 metadata.Add("category", "guidance");

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

To retrieve metadata, call one of the following methods:

- [GetProperties](#)
- [GetPropertiesAsync](#).

Then, read the values, as shown in the example below.

```
public static async Task ReadContainerMetadataAsync(BlobContainerClient container)
{
 try
 {
 var properties = await container.GetPropertiesAsync();

 // Enumerate the container's metadata.
 Console.WriteLine("Container metadata:");
 foreach (var metadataItem in properties.Value.Metadata)
 {
 Console.WriteLine($" \tKey: {metadataItem.Key}");
 Console.WriteLine($" \tValue: {metadataItem.Value}");
 }
 }
 catch (RequestFailedException e)
 {
 Console.WriteLine($"HTTP error code {e.Status}: {e.ErrorCode}");
 Console.WriteLine(e.Message);
 Console.ReadLine();
 }
}
```

## See also

- [Get started with Azure Blob Storage and .NET](#)
- [Get Container Properties operation](#)
- [Set Container Metadata operation](#)
- [Get Container Metadata operation](#)

# Upload a blob to Azure Storage by using the .NET client library

8/22/2022 • 3 minutes to read • [Edit Online](#)

You can upload a blob, open a blob stream and write to that, or upload large blobs in blocks.

## NOTE

The examples in this article assume that you've created a [BlobServiceClient](#) object by using the guidance in the [Get started with Azure Blob Storage and .NET](#) article. Blobs in Azure Storage are organized into containers. Before you can upload a blob, you must first create a container. To learn how to create a container, see [Create a container in Azure Storage with .NET](#).

To upload a blob by using a file path, a stream, a binary object or a text string, use either of the following methods:

- [Upload](#)
- [UploadAsync](#)

To open a stream in Blob Storage, and then write to that stream, use either of the following methods:

- [OpenWrite](#)
- [OpenWriteAsync](#)

## Upload by using a file path

The following example uploads a blob by using a file path:

```
public static async Task UploadFile
 (BlobContainerClient containerClient, string localFilePath)
{
 string fileName = Path.GetFileName(localFilePath);
 BlobClient blobClient = containerClient.GetBlobClient(fileName);

 await blobClient.UploadAsync(localFilePath, true);
}
```

## Upload by using a Stream

The following example uploads a blob by creating a [Stream](#) object, and then uploading that stream.

```
public static async Task UploadStream
 (BlobContainerClient containerClient, string localFilePath)
{
 string fileName = Path.GetFileName(localFilePath);
 BlobClient blobClient = containerClient.GetBlobClient(fileName);

 FileStream fileStream = File.OpenRead(localFilePath);
 await blobClient.UploadAsync(fileStream, true);
 fileStream.Close();
}
```

## Upload by using a BinaryData object

The following example uploads a [BinaryData](#) object.

```
public static async Task UploadBinary
 (BlobContainerClient containerClient, string localFilePath)
{
 string fileName = Path.GetFileName(localFilePath);
 BlobClient blobClient = containerClient.GetBlobClient(fileName);

 FileStream fileStream = File.OpenRead(localFilePath);
 BinaryReader reader = new BinaryReader(fileStream);

 byte[] buffer = new byte[fileStream.Length];

 reader.Read(buffer, 0, buffer.Length);

 BinaryData binaryData = new BinaryData(buffer);

 await blobClient.UploadAsync(binaryData, true);

 fileStream.Close();
}
```

## Upload a string

The following example uploads a string:

```
public static async Task UploadString
 (BlobContainerClient containerClient, string localFilePath)
{
 string fileName = Path.GetFileName(localFilePath);
 BlobClient blobClient = containerClient.GetBlobClient(fileName);

 await blobClient.UploadAsync(BinaryData.FromString("hello world"), overwrite: true);
}
```

## Upload with index tags

Blob index tags categorize data in your storage account using key-value tag attributes. These tags are automatically indexed and exposed as a searchable multi-dimensional index to easily find data. You can perform this task by adding tags to a [BlobUploadOptions](#) instance, and then passing that instance into the [UploadAsync](#) method.

The following example uploads a blob with three index tags.

```

public static async Task UploadBlobWithTags
 (BlobContainerClient containerClient, string localFilePath)
{
 string fileName = Path.GetFileName(localFilePath);
 BlobClient blobClient = containerClient.GetBlobClient(fileName);

 BlobUploadOptions options = new BlobUploadOptions();
 options.Tags = new Dictionary<string, string>
 {
 { "Sealed", "false" },
 { "Content", "image" },
 { "Date", "2020-04-20" }
 };

 await blobClient.UploadAsync(localFilePath, options);
}

```

## Upload to a stream in Blob Storage

You can open a stream in Blob Storage and write to that stream. The following example creates a zip file in Blob Storage and writes files to that file. Instead of building a zip file in local memory, only one file at a time is in memory.

```

public static async Task UploadToStream
 (BlobContainerClient containerClient, string localDirectoryPath)
{
 string zipFileName = Path.GetFileName
 (Path.GetDirectoryName(localDirectoryPath)) + ".zip";

 BlockBlobClient blockBlobClient =
 containerClient.GetBlockBlobClient(zipFileName);

 using (Stream stream = await blockBlobClient.OpenWriteAsync(true))
 {
 using (ZipArchive zip = new ZipArchive
 (stream, ZipArchiveMode.Create, leaveOpen: false))
 {
 foreach (var fileName in Directory.EnumerateFiles(localDirectoryPath))
 {
 using (var fileStream = File.OpenRead(fileName))
 {
 var entry = zip.CreateEntry(Path.GetFileName
 (fileName), CompressionLevel.Optimal);
 using (var innerFile = entry.Open())
 {
 await fileStream.CopyToAsync(innerFile);
 }
 }
 }
 }
 }
}

```

## Upload by staging blocks and then committing them

You can have greater control over how to divide our uploads into blocks by manually staging individual blocks of data. When all of the blocks that make up a blob are staged, you can commit them to Blob Storage. You can use this approach if you want to enhance performance by uploading blocks in parallel.

```

public static async Task UploadInBlocks
 (BlobContainerClient blobContainerClient, string localFilePath, int blockSize)
{
 string fileName = Path.GetFileName(localFilePath);
 BlockBlobClient blobClient = blobContainerClient.GetBlockBlobClient(fileName);

 FileStream fileStream = File.OpenRead(localFilePath);

 ArrayList blockIDArrayList = new ArrayList();

 byte[] buffer;

 var bytesLeft = (fileStream.Length - fileStream.Position);

 while (bytesLeft > 0)
 {
 if (bytesLeft >= blockSize)
 {
 buffer = new byte[blockSize];
 await fileStream.ReadAsync(buffer, 0, blockSize);
 }
 else
 {
 buffer = new byte[bytesLeft];
 await fileStream.ReadAsync(buffer, 0, Convert.ToInt32(bytesLeft));
 bytesLeft = (fileStream.Length - fileStream.Position);
 }

 using (var stream = new MemoryStream(buffer))
 {
 string blockID = Convert.ToBase64String
 (Encoding.UTF8.GetBytes(Guid.NewGuid().ToString()));

 blockIDArrayList.Add(blockID);

 await blobClient.StageBlockAsync(blockID, stream);
 }
 }

 bytesLeft = (fileStream.Length - fileStream.Position);
}

string[] blockIDArray = (string[])blockIDArrayList.ToArray(typeof(string));

await blobClient.CommitBlockListAsync(blockIDArray);
}

```

## See also

- [Manage and find Azure Blob data with blob index tags](#)
- [Use blob index tags to manage and find data on Azure Blob Storage](#)
- [Put Blob \(REST API\)](#)
- [Put Blob From URL \(REST API\)](#)

# Append data to a blob in Azure Storage using the .NET client library

8/22/2022 • 2 minutes to read • [Edit Online](#)

You can append data to a blob by creating an append blob. 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.

## NOTE

The examples in this article assume that you've created a [BlobServiceClient](#) object by using the guidance in the [Get started with Azure Blob Storage and .NET](#) article. Blobs in Azure Storage are organized into containers. Before you can upload a blob, you must first create a container. To learn how to create a container, see [Create a container in Azure Storage with .NET](#).

## Create an append blob and append data

Use these methods to create an append blob.

- [Create](#)
- [CreateAsync](#)
- [CreateIfNotExists](#)
- [CreateIfNotExistsAsync](#)

Use either of these methods to append data to that append blob:

- [AppendBlock](#)
- [AppendBlockAsync](#)

The maximum size in bytes of each append operation is defined by the [AppendBlobMaxAppendBlockBytes](#) property. The following example creates an append blob and appends log data to that blob. This example uses the [AppendBlobMaxAppendBlockBytes](#) property to determine whether multiple append operations are required.

```
public static async void AppendToBlob
 (BlobContainerClient containerClient, MemoryStream logEntryStream, string LogBlobName)
{
 AppendBlobClient appendBlobClient = containerClient.GetAppendBlobClient(LogBlobName);

 appendBlobClient.CreateIfNotExists();

 var maxBlockSize = appendBlobClient.AppendBlobMaxAppendBlockBytes;

 var buffer = new byte[maxBlockSize];

 if (logEntryStream.Length <= maxBlockSize)
 {
 appendBlobClient.AppendBlock(logEntryStream);
 }
 else
 {
 var bytesLeft = (logEntryStream.Length - logEntryStream.Position);

 while (bytesLeft > 0)
 {
 if (bytesLeft >= maxBlockSize)
 {
 buffer = new byte[maxBlockSize];
 await logEntryStream.ReadAsync
 (buffer, 0, maxBlockSize);
 }
 else
 {
 buffer = new byte[bytesLeft];
 await logEntryStream.ReadAsync
 (buffer, 0, Convert.ToInt32(bytesLeft));
 }

 appendBlobClient.AppendBlock(new MemoryStream(buffer));

 bytesLeft = (logEntryStream.Length - logEntryStream.Position);
 }
 }
}
```

## See also

- [Understanding block blobs, append blobs, and page blobs](#)
- [OpenWrite / OpenWriteAsync](#)
- [Append Block \(REST API\)](#)

# Download a blob in Azure Storage using the .NET client library

8/22/2022 • 2 minutes to read • [Edit Online](#)

You can download a blob by using any of the following methods:

- [DownloadTo](#)
- [DownloadToAsync](#)
- [DownloadContent](#) - [DownloadContentAsync](#)

You can also open a stream to read from a blob. The stream will only download the blob as the stream is read from. Use either of the following methods:

- [OpenRead](#)
- [OpenReadAsync](#)

## NOTE

The examples in this article assume that you've created a [BlobServiceClient](#) object by using the guidance in the [Get started with Azure Blob Storage and .NET](#) article.

## Download to a file path

The following example downloads a blob by using a file path:

```
public static async Task DownloadBlob(BlobClient blobClient, string localFilePath)
{
 await blobClient.DownloadToAsync(localFilePath);
}
```

## Download to a stream

The following example downloads a blob by creating a [Stream](#) object and then downloading to that stream.

```
public static async Task DownloadToStream(BlobClient blobClient, string localFilePath)
{
 FileStream fileStream = File.OpenWrite(localFilePath);
 await blobClient.DownloadToAsync(fileStream);
 fileStream.Close();
}
```

## Download to a string

The following example downloads a blob to a string. This example assumes that the blob is a text file.

```
public static async Task DownloadToText(BlobClient blobClient)
{
 BlobDownloadResult downloadResult = await blobClient.DownloadContentAsync();
 string downloadedData = downloadResult.Content.ToString();
 Console.WriteLine("Downloaded data:", downloadedData);
}
```

## Download from a stream

The following example downloads a blob by reading from a stream.

```
public static async Task DownloadfromStream(BlobClient blobClient, string localFilePath)
{
 using (var stream = await blobClient.OpenReadAsync())
 {
 FileStream fileStream = File.OpenWrite(localFilePath);
 await stream.CopyToAsync(fileStream);
 }
}
```

## See also

- [Get started with Azure Blob Storage and .NET](#)
- [DownloadStreaming / DownloadStreamingAsync](#)
- [Get Blob \(REST API\)](#)

# Copy a blob with Azure Storage using the .NET client library

8/22/2022 • 4 minutes to read • [Edit Online](#)

This article demonstrates how to copy a blob in an Azure Storage account. It also shows how to abort an asynchronous copy operation. The example code uses the Azure Storage client libraries.

## About copying blobs

When you copy a blob within the same storage account, it's a synchronous operation. When you copy across accounts it's an asynchronous operation.

The source blob for a copy operation may be a block blob, an append blob, a page blob, or a snapshot. If the destination blob already exists, it must be of the same blob type as the source blob. An existing destination blob will be overwritten.

The destination blob can't be modified while a copy operation is in progress. A destination blob can only have one outstanding copy operation. In other words, a blob can't be the destination for multiple pending copy operations.

The entire source blob or file is always copied. Copying a range of bytes or set of blocks is not supported.

When a blob is copied, its system properties are copied to the destination blob with the same values.

A copy operation can take any of the following forms:

- Copy a source blob to a destination blob with a different name. The destination blob can be an existing blob of the same blob type (block, append, or page), or can be a new blob created by the copy operation.
- Copy a source blob to a destination blob with the same name, effectively replacing the destination blob. Such a copy operation removes any uncommitted blocks and overwrites the destination blob's metadata.
- Copy a source file in the Azure File service to a destination blob. The destination blob can be an existing block blob, or can be a new block blob created by the copy operation. Copying from files to page blobs or append blobs is not supported.
- Copy a snapshot over its base blob. By promoting a snapshot to the position of the base blob, you can restore an earlier version of a blob.
- Copy a snapshot to a destination blob with a different name. The resulting destination blob is a writeable blob and not a snapshot.

## Copy a blob

To copy a blob, call one of the following methods:

- [StartCopyFromUri](#)
- [StartCopyFromUriAsync](#)

The `StartCopyFromUri` and `StartCopyFromUriAsync` methods return a `CopyFromUriOperation` object containing information about the copy operation.

The following code example gets a `BlobClient` representing a previously created blob and copies it to a new blob in the same container:

```

private static async Task CopyBlobAsync(BlobContainerClient container)
{
 try
 {
 // Get the name of the first blob in the container to use as the source.
 string blobName = container.GetBlobs().FirstOrDefault().Name;

 // Create a BlobClient representing the source blob to copy.
 BlobClient sourceBlob = container.GetBlobClient(blobName);

 // Ensure that the source blob exists.
 if (await sourceBlob.ExistsAsync())
 {
 // Lease the source blob for the copy operation
 // to prevent another client from modifying it.
 BlobLeaseClient lease = sourceBlob.GetBlobLeaseClient();

 // Specifying -1 for the lease interval creates an infinite lease.
 await lease.AcquireAsync(TimeSpan.FromSeconds(-1));

 // Get the source blob's properties and display the lease state.
 BlobProperties sourceProperties = await sourceBlob.GetPropertiesAsync();
 Console.WriteLine($"Lease state: {sourceProperties.LeaseState}");

 // Get a BlobClient representing the destination blob with a unique name.
 BlobClient destBlob =
 container.GetBlobClient(Guid.NewGuid() + "-" + sourceBlob.Name);

 // Start the copy operation.
 await destBlob.StartCopyFromUriAsync(sourceBlob.Uri);

 // Get the destination blob's properties and display the copy status.
 BlobProperties destProperties = await destBlob.GetPropertiesAsync();

 Console.WriteLine($"Copy status: {destProperties.CopyStatus}");
 Console.WriteLine($"Copy progress: {destProperties.CopyProgress}");
 Console.WriteLine($"Completion time: {destProperties.CopyCompletedOn}");
 Console.WriteLine($"Total bytes: {destProperties.ContentLength}");

 // Update the source blob's properties.
 sourceProperties = await sourceBlob.GetPropertiesAsync();

 if (sourceProperties.LeaseState == LeaseState.Leased)
 {
 // Break the lease on the source blob.
 await lease.BreakAsync();

 // Update the source blob's properties to check the lease state.
 sourceProperties = await sourceBlob.GetPropertiesAsync();
 Console.WriteLine($"Lease state: {sourceProperties.LeaseState}");
 }
 }
 }
 catch (RequestFailedException ex)
 {
 Console.WriteLine(ex.Message);
 Console.ReadLine();
 throw;
 }
}

```

## Abort a copy operation

Aborting a copy operation results in a destination blob of zero length. However, the metadata for the destination blob will have the new values copied from the source blob or set explicitly during the copy operation. To keep

the original metadata from before the copy, make a snapshot of the destination blob before calling one of the copy methods.

- [.NET v12 SDK](#)

Check the `BlobProperties.CopyStatus` property on the destination blob to get the status of the copy operation. The final blob will be committed when the copy completes.

When you abort a copy operation, the destination blob's copy status is set to [CopyStatus.Aborted](#).

The [AbortCopyFromUri](#) and [AbortCopyFromUriAsync](#) methods cancel an ongoing copy operation.

```
// Get the destination blob's properties to check the copy status.
BlobProperties destProperties = destBlob.GetProperties();

// Check the copy status. If the status is pending, abort the copy operation.
if (destProperties.CopyStatus == CopyStatus.Pending)
{
 await destBlob.AbortCopyFromUriAsync(destProperties.CopyId);
 Console.WriteLine($"Copy operation {destProperties.CopyId} has been aborted.");
}
```

## See also

- [Copy Blob](#)
- [Abort Copy Blob](#)
- [Get started with Azure Blob Storage and .NET](#)

# List blobs using the Azure Storage client library for .NET

8/22/2022 • 4 minutes to read • [Edit Online](#)

When you list blobs from your code, you can specify a number of options to manage how results are returned from Azure Storage. You can specify the number of results to return in each set of results, and then retrieve the subsequent sets. You can specify a prefix to return blobs whose names begin with that character or string. And you can list blobs in a flat listing structure, or hierarchically. A hierarchical listing returns blobs as though they were organized into folders.

## Understand blob listing options

To list the blobs in a storage account, call one of these methods:

- [BlobContainerClient.GetBlobs](#)
- [BlobContainerClient.GetBlobsAsync](#)
- [BlobContainerClient.GetBlobsByHierarchy](#)
- [BlobContainerClient.GetBlobsByHierarchyAsync](#)

### Manage how many results are returned

By default, a listing operation returns up to 5000 results at a time, but you can specify the number of results that you want each listing operation to return. The examples presented in this article show you how to return results in pages.

### Filter results with a prefix

To filter the list of blobs, specify a string for the `prefix` parameter. The prefix string can include one or more characters. Azure Storage then returns only the blobs whose names start with that prefix.

### Return metadata

You can return blob metadata with the results by specifying the **Metadata** value for the [BlobTraits](#) enumeration.

### Flat listing versus hierarchical listing

Blobs in Azure Storage are organized in a flat paradigm, rather than a hierarchical paradigm (like a classic file system). However, you can organize blobs into *virtual directories* in order to mimic a folder structure. A virtual directory forms part of the name of the blob and is indicated by the delimiter character.

To organize blobs into virtual directories, use a delimiter character in the blob name. The default delimiter character is a forward slash (/), but you can specify any character as the delimiter.

If you name your blobs using a delimiter, then you can choose to list blobs hierarchically. For a hierarchical listing operation, Azure Storage returns any virtual directories and blobs beneath the parent object. You can call the listing operation recursively to traverse the hierarchy, similar to how you would traverse a classic file system programmatically.

## Use a flat listing

By default, a listing operation returns blobs in a flat listing. In a flat listing, blobs are not organized by virtual directory.

The following example lists the blobs in the specified container using a flat listing, with an optional segment size

specified, and writes the blob name to a console window.

If you've enabled the hierarchical namespace feature on your account, directories are not virtual. Instead, they are concrete, independent objects. Therefore, directories appear in the list as zero-length blobs.

```
private static async Task ListBlobsFlatListing(BlobContainerClient blobContainerClient,
 int? segmentSize)
{
 try
 {
 // Call the listing operation and return pages of the specified size.
 var resultSegment = blobContainerClient.GetBlobsAsync()
 .AsPages(default, segmentSize);

 // Enumerate the blobs returned for each page.
 await foreach (Azure.Page<BlobItem> blobPage in resultSegment)
 {
 foreach (BlobItem blobItem in blobPage.Values)
 {
 Console.WriteLine("Blob name: {0}", blobItem.Name);
 }

 Console.WriteLine();
 }
 }
 catch (RequestFailedException e)
 {
 Console.WriteLine(e.Message);
 Console.ReadLine();
 throw;
 }
}
```

The sample output is similar to:

```
Blob name: FolderA/blob1.txt
Blob name: FolderA/blob2.txt
Blob name: FolderA/blob3.txt
Blob name: FolderA/FolderB/blob1.txt
Blob name: FolderA/FolderB/blob2.txt
Blob name: FolderA/FolderB/blob3.txt
Blob name: FolderA/FolderB/FolderC/blob1.txt
Blob name: FolderA/FolderB/FolderC/blob2.txt
Blob name: FolderA/FolderB/FolderC/blob3.txt
```

## Use a hierarchical listing

When you call a listing operation hierarchically, Azure Storage returns the virtual directories and blobs at the first level of the hierarchy.

To list blobs hierarchically, call the [BlobContainerClient.GetBlobsByHierarchy](#), or the [BlobContainerClient.GetBlobsByHierarchyAsync](#) method.

The following example lists the blobs in the specified container using a hierarchical listing, with an optional segment size specified, and writes the blob name to the console window.

```

private static async Task ListBlobsHierarchicalListing(BlobContainerClient container,
 string prefix,
 int? segmentSize)
{
 try
 {
 // Call the listing operation and return pages of the specified size.
 var resultSegment = container.GetBlobsByHierarchyAsync(prefix:prefix, delimiter:"/")
 .AsPages(default, segmentSize);

 // Enumerate the blobs returned for each page.
 await foreach (Azure.Page<BlobHierarchyItem> blobPage in resultSegment)
 {
 // A hierarchical listing may return both virtual directories and blobs.
 foreach (BlobHierarchyItem blobhierarchyItem in blobPage.Values)
 {
 if (blobhierarchyItem.IsPrefix)
 {
 // Write out the prefix of the virtual directory.
 Console.WriteLine("Virtual directory prefix: {0}", blobhierarchyItem.Prefix);

 // Call recursively with the prefix to traverse the virtual directory.
 await ListBlobsHierarchicalListing(container, blobhierarchyItem.Prefix, null);
 }
 else
 {
 // Write out the name of the blob.
 Console.WriteLine("Blob name: {0}", blobhierarchyItem.Blob.Name);
 }
 }

 Console.WriteLine();
 }
 }
 catch (RequestFailedException e)
 {
 Console.WriteLine(e.Message);
 Console.ReadLine();
 throw;
 }
}

```

The sample output is similar to:

```

Virtual directory prefix: FolderA/
Blob name: FolderA/blob1.txt
Blob name: FolderA/blob2.txt
Blob name: FolderA/blob3.txt

Virtual directory prefix: FolderA/FolderB/
Blob name: FolderA/FolderB/blob1.txt
Blob name: FolderA/FolderB/blob2.txt
Blob name: FolderA/FolderB/blob3.txt

Virtual directory prefix: FolderA/FolderB/FolderC/
Blob name: FolderA/FolderB/FolderC/blob1.txt
Blob name: FolderA/FolderB/FolderC/blob2.txt
Blob name: FolderA/FolderB/FolderC/blob3.txt

```

#### **NOTE**

Blob snapshots cannot be listed in a hierarchical listing operation.

## List blob versions or snapshots

To list blob versions or snapshots, specify the [BlobStates](#) parameter with the [Version](#) or [Snapshot](#) field. Versions and snapshots are listed from oldest to newest.

The following code example shows how to list blob versions.

```
private static void ListBlobVersions(BlobContainerClient blobContainerClient,
 string blobName)
{
 // Call the listing operation, specifying that blob versions are returned.
 // Use the blob name as the prefix.
 var blobVersions = blobContainerClient.GetBlobs
 (BlobTraits.None, BlobStates.Version, prefix: blobName)
 .OrderByDescending(version => version.VersionId);

 // Construct the URI for each blob version.
 foreach (var version in blobVersions)
 {
 BlobUriBuilder blobUriBuilder = new BlobUriBuilder(blobContainerClient.Uri)
 {
 BlobName = version.Name,
 VersionId = version.VersionId
 };

 if ((bool)version.IsLatestVersion.GetValueOrDefault())
 {
 Console.WriteLine("Current version: {0}", blobUriBuilder);
 }
 else
 {
 Console.WriteLine("Previous version: {0}", blobUriBuilder);
 }
 }
}
```

## Next steps

- [List Blobs](#)
- [Enumerating Blob Resources](#)
- [Blob versioning](#)

# Delete and restore a blob in your Azure Storage account using the .NET client library

8/22/2022 • 2 minutes to read • [Edit Online](#)

This article shows how to delete blobs with the [Azure Storage client library for .NET](#). If you've enabled blob soft delete, you can restore deleted blobs.

## Delete a blob

To delete a blob, call either of these methods:

- [Delete](#)
- [DeleteAsync](#)
- [DeleteIfExists](#)
- [DeleteIfExistsAsync](#)

The following example deletes a blob.

```
public static async Task DeleteBlob(BlobClient blob)
{
 await blob.DeleteAsync();
}
```

## Restore a deleted blob

Blob soft delete protects an individual blob and its versions, snapshots, and metadata 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 the blob to its state at deletion. After the retention period has expired, the blob is permanently deleted. For more information about blob soft delete, see [Soft delete for blobs](#).

You can use the Azure Storage client libraries to restore a soft-deleted blob or snapshot.

### **Restore soft-deleted objects when versioning is disabled**

To restore deleted blobs when versioning is not enabled, call either of the following methods:

- [Undelete](#)
- [UndeleteAsync](#)

These methods restore soft-deleted blobs and any deleted snapshots associated with them. Calling either of these methods for a blob that has not been deleted has no effect. The following example restores all soft-deleted blobs and their snapshots in a container:

```
public static async Task UndeleteBlobs(BlobContainerClient container)
{
 foreach (BlobItem blob in container.GetBlobs(BlobTraits.None, BlobStates.Deleted))
 {
 await container.GetBlockBlobClient(blob.Name).UndeleteAsync();
 }
}
```

To restore a specific soft-deleted snapshot, first call the [Undelete](#) or [UndeleteAsync](#) on the base blob, then copy

the desired snapshot over the base blob. The following example restores a block blob to the most recently generated snapshot:

```
public static async Task RestoreSnapshots(BlobContainerClient container, BlobClient blob)
{
 // Restore the deleted blob.
 await blob.UndeleteAsync();

 // List blobs in this container that match prefix.
 // Include snapshots in listing.
 Pageable<BlobItem> blobItems = container.GetBlobs
 (BlobTraits.None, BlobStates.Snapshots, prefix: blob.Name);

 // Get the URI for the most recent snapshot.
 BlobUriBuilder blobSnapshotUri = new BlobUriBuilder(blob.Uri)
 {
 Snapshot = blobItems
 .OrderByDescending(snapshot => snapshot.Snapshot)
 .ElementAtOrDefault(0)?.Snapshot
 };

 // Restore the most recent snapshot by copying it to the blob.
 blob.StartCopyFromUri(blobSnapshotUri.ToUri());
}
```

#### Restore soft-deleted blobs when versioning is enabled

To restore a soft-deleted blob when versioning is enabled, copy a previous version over the base blob. You can use either of the following methods:

- [StartCopyFromUri](#)
- [StartCopyFromUriAsync](#)

```
public static void RestoreBlobsWithVersioning(BlobContainerClient container, BlobClient blob)
{
 // List blobs in this container that match prefix.
 // Include versions in listing.
 Pageable<BlobItem> blobItems = container.GetBlobs
 (BlobTraits.None, BlobStates.Version, prefix: blob.Name);

 // Get the URI for the most recent version.
 BlobUriBuilder blobVersionUri = new BlobUriBuilder(blob.Uri)
 {
 VersionId = blobItems
 .OrderByDescending(version => version.VersionId)
 .ElementAtOrDefault(0)?.VersionId
 };

 // Restore the most recently generated version by copying it to the base blob.
 blob.StartCopyFromUri(blobVersionUri.ToUri());
}
```

## See also

- [Get started with Azure Blob Storage and .NET](#)
- [Delete Blob \(REST API\)](#)
- [Soft delete for blobs](#)
- [Undelete Blob \(REST API\)](#)

# Use blob index tags to manage and find data in Azure Blob Storage (.NET)

8/22/2022 • 2 minutes to read • [Edit Online](#)

Blob index tags categorize data in your storage account using key-value tag attributes. These tags are automatically indexed and exposed as a searchable multi-dimensional index to easily find data. This article shows you how to set, get, and find data using blob index tags.

To learn more about this feature along with known issues and limitations, see [Manage and find Azure Blob data with blob index tags](#).

## Set and retrieve index tags

You can set and get index tags if your code has authorized access by using an account key or if your code uses a security principal that has been given the appropriate role assignments. For more information, see [Manage and find Azure Blob data with blob index tags](#).

### Set tags

You can set tags by using either of the following methods:

- [SetTags](#)
- [SetTagsAsync](#)

The following example performs this task.

```
public static async Task SetTags(BlobClient blobClient)
{
 Dictionary<string, string> tags =
 new Dictionary<string, string>
 {
 { "Sealed", "false" },
 { "Content", "image" },
 { "Date", "2020-04-20" }
 };

 await blobClient.SetTagsAsync(tags);
}
```

You can delete all tags by passing an empty [Dictionary] into the [SetTags](#) or [SetTagsAsync](#) method as shown in the following example.

```
Dictionary<string, string> noTags = new Dictionary<string, string>();
await blobClient.SetTagsAsync(noTags);
```

### RELATED ARTICLES

[Manage and find Azure Blob data with blob index tags](#)

[Set Blob Tags \(REST API\)](#)

[Get tags](#)

You can get tags by using either of the following methods:

- [GetTags](#)
- [GetTagsAsync](#)

The following example performs this task.

```
public static async Task GetTags(BlobClient blobClient)
{
 Response<GetBlobTagResult> tagsResponse = await blobClient.GetTagsAsync();

 foreach (KeyValuePair<string, string> tag in tagsResponse.Value.Tags)
 {
 Console.WriteLine($"{tag.Key}={tag.Value}");
 }
}
```

## Filter and find data with blob index tags

You can use index tags to find and filter data if your code has authorized access by using an account key or if your code uses a security principal that has been given the appropriate role assignments. For more information, see [Manage and find Azure Blob data with blob index tags](#).

### NOTE

You can't use index tags to retrieve previous versions. Tags for previous versions aren't passed to the blob index engine. For more information, see [Conditions and known issues](#).

You can find data by using either of the following methods:

- [FindBlobsByTags](#)
- [FindBlobsByTagsAsync](#)

The following example finds all blobs tagged with a date that falls between a specific range.

```
public static async Task FindBlobsbyTags(BlobServiceClient serviceClient)
{
 string query = @"""Date"" >= '2020-04-20' AND ""Date"" <= '2020-04-30';

 // Find Blobs given a tags query
 Console.WriteLine("Find Blob by Tags query: " + query + Environment.NewLine);

 List<TaggedBlobItem> blobs = new List<TaggedBlobItem>();
 await foreach (TaggedBlobItem taggedBlobItem in serviceClient.FindBlobsByTagsAsync(query))
 {
 blobs.Add(taggedBlobItem);
 }

 foreach (var filteredBlob in blobs)
 {

 Console.WriteLine($"BlobIndex result: ContainerName= {filteredBlob.BlobContainerName}, " +
 $"BlobName= {filteredBlob.BlobName}");
 }
}
```

## See also

- [Manage and find Azure Blob data with blob index tags](#)
- [Get Blob Tags \(REST API\)](#)
- [Find Blobs by Tags \(REST API\)](#)

# Manage blob properties and metadata with .NET

8/22/2022 • 4 minutes to read • [Edit Online](#)

In addition to the data they contain, blobs support system properties and user-defined metadata. This article shows how to manage system properties and user-defined metadata with the [Azure Storage client library for .NET](#).

## 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.

### NOTE

Blob index tags also provide the ability to store arbitrary user-defined key/value attributes alongside an Azure Blob storage resource. While similar to metadata, only blob index tags are automatically indexed and made searchable by the native blob service. Metadata cannot be indexed and queried unless you utilize a separate service such as Azure Search.

To learn more about this feature, see [Manage and find data on Azure Blob storage with blob index \(preview\)](#).

## Set and retrieve properties

The following code example sets the `ContentType` and `ContentLanguage` system properties on a blob.

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 are not being updated.

```

public static async Task SetBlobPropertiesAsync(BlobClient blob)
{
 Console.WriteLine("Setting blob properties...");

 try
 {
 // Get the existing properties
 BlobProperties properties = await blob.GetPropertiesAsync();

 BlobHttpHeaders headers = new BlobHttpHeaders
 {
 // Set the MIME ContentType every time the properties
 // are updated or the field will be cleared
 ContentType = "text/plain",
 ContentLanguage = "en-us",

 // Populate remaining headers with
 // the pre-existing properties
 CacheControl = properties.CacheControl,
 ContentDisposition = properties.ContentDisposition,
 ContentEncoding = properties.ContentEncoding,
 ContentHash = properties.ContentHash
 };

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

```

The following code example gets a blob's system properties and displays some of the values.

```

private static async Task GetBlobPropertiesAsync(BlobClient blob)
{
 try
 {
 // Get the blob properties
 BlobProperties properties = await blob.GetPropertiesAsync();

 // Display some of the blob's property values
 Console.WriteLine($" ContentLanguage: {properties.ContentLanguage}");
 Console.WriteLine($" ContentType: {properties.ContentType}");
 Console.WriteLine($" CreatedOn: {properties.CreatedOn}");
 Console.WriteLine($" LastModified: {properties.LastModified}");
 }
 catch (RequestFailedException e)
 {
 Console.WriteLine($"HTTP error code {e.Status}: {e.ErrorCode}");
 Console.WriteLine(e.Message);
 Console.ReadLine();
 }
}

```

## 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](#)

Metadata name/value pairs are valid HTTP headers and should adhere to all restrictions governing HTTP headers. Metadata names must be valid HTTP header names and valid C# identifiers, may contain only ASCII characters, and should be treated as case-insensitive. [Base64-encode](#) or [URL-encode](#) metadata values containing non-ASCII characters.

The name of your metadata must conform to the naming conventions for C# identifiers. Metadata names maintain the case used when they were created, but are case-insensitive when set or read. If two or more metadata headers using the same name are submitted for a resource, Azure Blob storage returns HTTP error code 400 (Bad Request).

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();
 }
}
```

The following code example reads the metadata on a blob.

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](#) methods retrieve blob properties and metadata in a single call. This is different from the REST APIs which require separate calls to [Get Blob Properties](#) and [Get Blob Metadata](#).

```
public static async Task ReadBlobMetadataAsync(BlobClient blob)
{
 try
 {
 // Get the blob's properties and metadata.
 BlobProperties properties = await blob.GetPropertiesAsync();

 Console.WriteLine("Blob metadata:");

 // Enumerate the blob's metadata.
 foreach (var metadataItem in properties.Metadata)
 {
 Console.WriteLine($" \tKey: {metadataItem.Key}");
 Console.WriteLine($" \tValue: {metadataItem.Value}");
 }
 }
 catch (RequestFailedException e)
 {
 Console.WriteLine($"HTTP error code {e.Status}: {e.ErrorCode}");
 Console.WriteLine(e.Message);
 Console.ReadLine();
 }
}
```

## See also

- [Set Blob Properties operation](#)
- [Get Blob Properties operation](#)
- [Set Blob Metadata operation](#)
- [Get Blob Metadata operation](#)

# Create and manage blob or container leases with .NET

8/22/2022 • 2 minutes to read • [Edit Online](#)

A lease establishes and manages a lock on a container or the blobs in a container. You can use the .NET client library to acquire, renew, release and break leases. To learn more about leasing blobs or containers, see [Lease Container](#) or [Lease Blobs](#).

## Acquire a lease

When you acquire a lease, you'll obtain a lease ID that your code can use to operate on the blob or container. To acquire a lease, create an instance of the [BlobLeaseClient](#) class, and then use either of these methods:

- [Acquire](#)
- [AcquireAsync](#)

The following example acquires a 30 second lease for a container.

```
public static async Task AcquireLease(BlobContainerClient containerClient)
{
 BlobLeaseClient blobLeaseClient = containerClient.GetBlobLeaseClient();

 TimeSpan ts = new TimeSpan(0, 0, 0, 30);
 Response<BlobLease> blobLeaseResponse = await blobLeaseClient.AcquireAsync(ts);

 Console.WriteLine("Blob Lease Id:" + blobLeaseResponse.Value.LeaseId);
 Console.WriteLine("Remaining Lease Time: " + blobLeaseResponse.Value.LeaseTime);
}
```

## Renew a lease

If your lease expires, you can renew it. To renew a lease, use either of the following methods of the [BlobLeaseClient](#) class:

- [Renew](#)
- [RenewAsync](#)

Specify the lease ID by setting the [IfMatch](#) property of a [RequestConditions](#) instance.

The following example renews a lease for a blob.

```
public static async Task RenewLease(BlobClient blobClient, string leaseID)
{
 BlobLeaseClient blobLeaseClient = blobClient.GetBlobLeaseClient();
 RequestConditions requestConditions = new RequestConditions();
 requestConditions.IfMatch = new ETag(leaseID);
 await blobLeaseClient.RenewAsync();
}
```

## Release a lease

You can either wait for a lease to expire or explicitly release it. When you release a lease, other clients can obtain

a lease. You can release a lease by using either of these methods of the [BlobLeaseClient](#) class.

- [Release](#)
- [ReleaseAsync](#)

The following example releases the lease on a container.

```
public static async Task ReleaseLease(BlobContainerClient containerClient)
{
 BlobLeaseClient blobLeaseClient = containerClient.GetBlobLeaseClient();
 await blobLeaseClient.ReleaseAsync();
}
```

## Break a lease

When you break a lease, the lease ends, but other clients can't acquire a lease until the lease period expires. You can break a lease by using either of these methods:

- [Break](#)
- [BreakAsync](#);

The following example breaks the lease on a blob.

```
public static async Task BreakLease(BlobClient blobClient)
{
 BlobLeaseClient blobLeaseClient = blobClient.GetBlobLeaseClient();
 await blobLeaseClient.BreakAsync();
}
```

## See also

- [Get started with Azure Blob Storage and .NET](#)
- [Managing Concurrency in Blob storage](#)

# Get started with Azure Blob Storage and JavaScript

8/22/2022 • 6 minutes to read • [Edit Online](#)

This article shows you how to connect to Azure Blob Storage by using the Azure Blob Storage client library v12 for JavaScript. Once connected, your code can operate on containers, blobs, and features of the Blob Storage service.

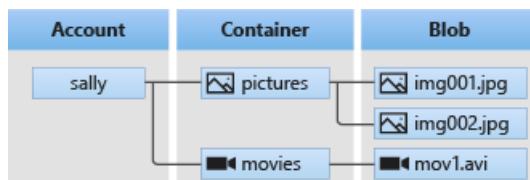
The [sample code snippets](#) are available in GitHub as runnable Node.js files.

[Package \(npm\)](#) | [Samples](#) | [API reference](#) | [Library source code](#) | [Give Feedback](#)

## SDK Objects for service, container, and blob

The [BlobServiceClient](#) object is the top object in the SDK. This client allows you to manipulate the service, containers and blobs. From the BlobServiceClient, you can get to the ContainerClient. The [ContainerClient](#) object allows you to interact with a container and its blobs. The [BlobClient](#) allows you to manipulate blobs.

CLIENT	ALLOWS ACCESS TO	ACCESSED
Account: <a href="#">BlobServiceClient</a>	Controls your service resource, provides access to container and blobs.	Directly from SDK via require statement.
Container: <a href="#">ContainerClient</a>	Controls a specific container, provides access to blobs.	Directly from SDK via require statement or from <a href="#">BlobServiceClient</a> .
Blob: <a href="#">BlobClient</a>	Access to a blob of any kind: <a href="#">block</a> , <a href="#">append</a> , <a href="#">page</a> .	Directly from SDK via require statement or from <a href="#">ContainerClient</a> .



## Prerequisites

- Azure subscription - [create one for free](#)
- Azure storage account - [create a storage account](#)
- [Node.js LTS](#)
- Optionally, you need [bundling tools](#) if you're developing for a web client.

## Set up your project

1. Open a command prompt and change into your project folder:

```
cd YOUR-DIRECTORY
```

2. If you don't have a `package.json` file already in your directory, initialize the project to create the file:

```
npm init -y
```

3. Install the Azure Blob Storage client library for JavaScript:

```
npm install @azure/storage-blob
```

4. If you want to connect with managed identity, install the Azure Identity client library for JavaScript:

```
npm install @azure/identity
```

5. In your `index.js` file, add the package:

```
const { BlobServiceClient, StorageSharedKeyCredential } = require('@azure/storage-blob');

// optional but recommended - connect with managed identity (Azure AD)
const { DefaultAzureCredential } = require('@azure/identity');
```

## Connect with Azure AD

Azure Active Directory (Azure AD) provides the most secure connection by managing the connection identity ([managed identity](#)). This functionality allows you to develop code that doesn't require any secrets (keys or connection strings) stored in the code or environment. Managed identity requires [setup](#) for any identities such as developer (personal) or cloud (hosting) environments. You need to complete the setup before using the code in this section.

After you complete the setup, your Storage resource needs to have one or more of the following roles assigned to the identity resource you plan to connect with:

- A [data access](#) role - such as:
  - **Storage Blob Data Reader**
  - **Storage Blob Data Contributor**
- A [resource](#) role - such as:
  - **Reader**
  - **Contributor**

To authorize with Azure AD, you'll need to use an Azure credential. Which type of credential you need depends on where your application runs. Use this table as a guide.

WHERE THE APPLICATION RUNS	SECURITY PRINCIPAL	GUIDANCE
Local machine (developing and testing)	User identity or service principal	<a href="#">Use the Azure Identity library to get an access token for authorization</a>
Azure	Managed identity	<a href="#">Authorize access to blob data with managed identities for Azure resources</a>
Servers or clients outside of Azure	Service principal	<a href="#">Authorize access to blob or queue data from a native or web application</a>

Create a [DefaultAzureCredential](#) instance. Use that object to create a [BlobServiceClient](#).

```

const { BlobServiceClient } = require('@azure/storage-blob');
const { DefaultAzureCredential } = require('@azure/identity');
require('dotenv').config()

const accountName = process.env.AZURE_STORAGE_ACCOUNT_NAME;
if (!accountName) throw Error('Azure Storage accountName not found');

const blobServiceClient = new BlobServiceClient(
 `https://${accountName}.blob.core.windows.net`,
 new DefaultAzureCredential()
);

async function main(){

 // this call requires Reader role on the identity
 const serviceGetPropertiesResponse = await blobServiceClient.getProperties();
 console.log(`"${JSON.stringify(serviceGetPropertiesResponse)}"`);

}

main()
 .then(() => console.log(`done`))
 .catch((ex) => console.log(`error: ${ex.message}`));

```

If you plan to deploy the application to servers and clients that run outside of Azure, you can obtain an OAuth token by using other classes in the [Azure Identity client library for JavaScript](#) which derive from the [TokenCredential](#) class.

## Connect with an account name and key

Create a [StorageSharedKeyCredential](#) by using the storage account name and account key. Then use the [StorageSharedKeyCredential](#) to initialize a [BlobServiceClient](#).

```

const { BlobServiceClient, StorageSharedKeyCredential } = require('@azure/storage-blob');
require('dotenv').config()

const accountName = process.env.AZURE_STORAGE_ACCOUNT_NAME;
const accountKey = process.env.AZURE_STORAGE_ACCOUNT_KEY;
if (!accountName) throw Error('Azure Storage accountName not found');
if (!accountKey) throw Error('Azure Storage accountKey not found');

const sharedKeyCredential = new StorageSharedKeyCredential(accountName, accountKey);

const blobServiceClient = new BlobServiceClient(
 `https://${accountName}.blob.core.windows.net`,
 sharedKeyCredential
);

async function main(){
 const serviceGetPropertiesResponse = await blobServiceClient.getProperties();
 console.log(`"${JSON.stringify(serviceGetPropertiesResponse)}"`);
}

main()
 .then(() => console.log(`done`))
 .catch((ex) => console.log(ex.message));

```

For information about how to obtain account keys and best practice guidelines for properly managing and safeguarding your keys, see [Manage storage account access keys](#).

## Connect with a connection string

Create a [BlobServiceClient](#) by using a connection string.

```
const { BlobServiceClient } = require('@azure/storage-blob');
require('dotenv').config()

const connString = process.env.AZURE_STORAGE_CONNECTION_STRING;
if (!connString) throw Error('Azure Storage Connection string not found');

const blobServiceClient = BlobServiceClient.fromConnectionString(connString);

async function main(){
 const serviceGetPropertiesResponse = await blobServiceClient.getProperties();
 console.log(`[${JSON.stringify(serviceGetPropertiesResponse)}]`);
}

main()
 .then(() => console.log(`done`))
 .catch((ex) => console.log(ex.message));
```

For information about how to obtain account keys and best practice guidelines for properly managing and safeguarding your keys, see [Manage storage account access keys](#).

## Object Authorization with a SAS token

Create a Uri to your resource by using the blob service endpoint and SAS token. Then, create a [BlobServiceClient](#) with the Uri.

```
const { BlobServiceClient } = require('@azure/storage-blob');
require('dotenv').config()

const accountName = process.env.AZURE_STORAGE_ACCOUNT_NAME;
const sasToken = process.env.AZURE_STORAGE_SAS_TOKEN;
if (!accountName) throw Error('Azure Storage accountName not found');
if (!sasToken) throw Error('Azure Storage accountKey not found');

const blobServiceUri = `https://${accountName}.blob.core.windows.net`;

const blobServiceClient = new BlobServiceClient(
 `${blobServiceUri}${sasToken}`,
 null
);

async function main(){
 const serviceGetPropertiesResponse = await blobServiceClient.getProperties();
 console.log(`[${JSON.stringify(serviceGetPropertiesResponse)}]`);
}

main()
 .then(() => console.log(`done`))
 .catch((ex) => console.log(`error: ${ex.message}`));
```

To generate and manage SAS tokens, see any of these articles:

- [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#)
- [Create a service SAS for a container or blob](#)

## Connect anonymously

If you explicitly enable anonymous access, then you can connect to Blob Storage without authorization for your request. You can create a new BlobServiceClient object for anonymous access by providing the Blob storage

endpoint for the account. This requires you to know the account and container names. To learn how to enable anonymous access, see [Configure anonymous public read access for containers and blobs](#).

```
const { BlobServiceClient, AnonymousCredential } = require('@azure/storage-blob');
require('dotenv').config()

const accountName = process.env.AZURE_STORAGE_ACCOUNT_NAME;
if (!accountName) throw Error('Azure Storage accountName not found');

const blobServiceUri = `https://${accountName}.blob.core.windows.net`;

const blobServiceClient = new BlobServiceClient(
 blobServiceUri,
 new AnonymousCredential()
);

async function getContainerProperties(){

 // Access level: 'container'
 const containerName = `blob-storage-dev-guide-1`;

 const containerClient = blobServiceClient.getContainerClient(containerName);
 const containerProperties = await containerClient.getProperties();
 console.log(JSON.stringify(containerProperties));

}

getContainerProperties()
 .then(() => console.log(`done`))
 .catch((ex) => console.log(`error: ${ex.message}`));

```

Each type of resource is represented by one or more associated JavaScript clients:

CLASS	DESCRIPTION
<a href="#">BlobServiceClient</a>	Represents the Blob Storage endpoint for your storage account.
<a href="#">ContainerClient</a>	Allows you to manipulate Azure Storage containers and their blobs.
<a href="#">BlobClient</a>	Allows you to manipulate Azure Storage blobs.

The following guides show you how to use each of these clients to build your application. The [sample code](#) shown is this guide is available on GitHub.

GUIDE	DESCRIPTION
<a href="#">Create a container</a>	Create containers.
<a href="#">Delete and restore containers</a>	Delete containers, and if soft-delete is enabled, restore deleted containers.
<a href="#">List containers</a>	List containers in an account and the various options available to customize a listing.
<a href="#">Manage properties and metadata</a>	Get and set properties and metadata for containers.

GUIDE	DESCRIPTION
<a href="#">Upload blobs</a>	Learn how to upload blobs by using strings, streams, file paths, and other methods.
<a href="#">Download blobs</a>	Download blobs by using strings, streams, and file paths.
<a href="#">Copy blobs</a>	Copy a blob from one account to another account.
<a href="#">List blobs</a>	List blobs in different ways.
<a href="#">Delete and restore</a>	Delete blobs, and if soft-delete is enabled, restore deleted blobs.
<a href="#">Find blobs using tags</a>	Set and retrieve indexed tags then use tags to find blobs.
<a href="#">Manage properties and metadata</a>	Get all system properties and set HTTP properties and metadata for blobs.

## See also

- [Package \(npm\)](#)
- [Samples](#)
- [API reference](#)
- [Library source code](#)
- [Give Feedback](#)

# Create and use account SAS tokens with Azure Blob Storage and JavaScript

8/22/2022 • 3 minutes to read • [Edit Online](#)

This article shows you how to create and use account SAS tokens to use the Azure Blob Storage client library v12 for JavaScript. Once connected, your code can operate on containers, blobs, and features of the Blob Storage service.

The [sample code snippets](#) are available in GitHub as runnable Node.js files.

[Package \(npm\)](#) | [Samples](#) | [API reference](#) | [Library source code](#) | [Give Feedback](#)

## Account SAS tokens

An **account SAS token** is one [type of SAS token](#) for access delegation provided by Azure Storage. An account SAS token provides access to Azure Storage. The token is only as restrictive as you define it when creating it. Because anyone with the token can use it to access your Storage account, you should define the token with the most restrictive permissions that still allow the token to complete the required tasks.

[Best practices for token](#) creation include limiting permissions:

- Services: blob, file, queue, table
- Resource types: service, container, or object
- Permissions such as create, read, write, update, and delete

## Add required dependencies to your application

Include the required dependencies to create an account SAS token.

```
const {
 BlobServiceClient,
 generateAccountSASQueryParameters,
 AccountSASPermissions,
 AccountSASServices,
 AccountSASResourceTypes,
 StorageSharedKeyCredential,
 SASProtocol
} = require('@azure/storage-blob');
require('dotenv').config()
```

## Get environment variables to create shared key credential

Use the Blob Storage account name and key to create a [StorageSharedKeyCredential](#). This key is required to create the SAS token and to use the SAS token.

Create a [StorageSharedKeyCredential](#) by using the storage account name and account key. Then use the `StorageSharedKeyCredential` to initialize a [BlobServiceClient](#).

```

const constants = {
 accountName: process.env.AZURE_STORAGE_ACCOUNT_NAME,
 accountKey: process.env.AZURE_STORAGE_ACCOUNT_KEY
};
const sharedKeyCredential = new StorageSharedKeyCredential(
 constants.accountName,
 constants.accountKey
);

```

## Async operation boilerplate

The remaining sample code snippets assume the following async boilerplate code for Node.js.

```

async function main() {

 const sasToken = await createAccountSas();

 await useSasToken(sasToken);
}

main()
 .then(() => {
 console.log(`done`);
 })
 .catch((ex) => {
 console.log(`Error: ${ex.message}`);
 });

```

## Create SAS token

Because this token can be used with blobs, queues, tables, and files, some of the settings are more broad than just blob options.

1. Create the options object.

The scope of the abilities of a SAS token is defined by the [AccountSASSignatureValues](#).

Use the following helper functions provided by the SDK to create the correct value types for the values:

- [AccountSASServices.parse\("btqf"\).toString\(\)](#):
  - b: blob
  - t: table
  - q: query
  - f: file
- [resourceTypes: AccountSASResourceTypes.parse\("sco"\).toString\(\)](#)
  - s: service
  - c: container - such as blob container, table or queue
  - o: object - blob, row, message
- [permissions: AccountSASPermissions.parse\("rwdlacupi"\)](#)
  - r: read
  - w: write
  - d: delete
  - l: list
  - f: filter
  - a: add

- c: create
- u: update
- t: tag access
- p: process - such as process messages in a queue
- i: set immutability policy

2. Pass the object to the [generateAccountSASQueryParameters](#) function, along with the [SharedKeyCredential](#), to create the SAS token.

Before returning the SAS token, prepend the query string delimiter, `?` .

```
async function createAccountSas() {

 const sasOptions = {

 services: AccountSASServices.parse("btqf").toString(), // blobs, tables, queues,
files
 resourceTypes: AccountSASResourceTypes.parse("sco").toString(), // service, container, object
 permissions: AccountSASPermissions.parse("rwdlacupi"), // permissions
 protocol: SASProtocol.Https,
 startsOn: new Date(),
 expiresOn: new Date(new Date().valueOf() + (10 * 60 * 1000)), // 10 minutes
 };

 const sasToken = generateAccountSASQueryParameters(
 sasOptions,
 sharedKeyCredential
).toString();

 console.log(`sasToken = ${sasToken}\n`);

 // prepend sasToken with `?`
 return (sasToken[0] === '?') ? sasToken : `?${sasToken}`;
}
```

3. Secure the SAS token until it is used.

## Use Blob service with account SAS token

To use the account SAS token, you need to combine it with the account name to create the URI. Pass the URI to create the blobServiceClient. Once you have the blobServiceClient, you can use that client to access your Blob service.

```
// example function of using sasToken
// sasToken is prepended with `?`
async function useSasToken(sasToken) {

 // Use SAS token to create authenticated connection to Blob Service
 const blobServiceClient = new BlobServiceClient(
 `https://${constants.accountName}.blob.core.windows.net${sasToken}`
);

 // Get Blob Service properties
 const blobServicePropertiesResponse = await blobServiceClient.getProperties();

 // Display Blob Service properties
 console.log(`Success: Properties ${JSON.stringify(blobServicePropertiesResponse)}\n`);
}
```

## See also

- [Types of SAS tokens](#)
- [How a shared access signature works](#)
- [API reference](#)
- [Library source code](#)
- [Give Feedback](#)

# Create a user delegation SAS token with Azure Blob Storage and JavaScript

8/22/2022 • 7 minutes to read • [Edit Online](#)

This article shows you how to create a user delegation SAS token in the Azure Blob Storage client library v12 for JavaScript. A [user delegation SAS](#), introduced with version 2018-11-09, is secured with Azure AD credentials and is supported for the Blob service only to:

- Grant access to an existing [container](#).
- Grant access to create, use, and delete [blobs](#).

To create a user delegation SAS, a client must have permissions to call the [blobServiceClient.getUserDelegationKey](#) operation. The key returned by this operation is used to sign the user delegation SAS. The security principal that calls this operation must be assigned an RBAC role that includes the Microsoft.Storage/storageAccounts/blobServices/generateUserDelegationKey/action.

The permissions granted to a client who possesses the SAS are the intersection of the permissions that were granted to the security principal that requested the user delegation key and the permissions that were granted to the resource on the SAS token in the signed [permissions](#) (sp) field. If a permission that's granted to the security principal via RBAC isn't also granted on the SAS token, that permission isn't granted to the client who attempts to use the SAS to access the resource.

The [sample code snippets](#) are available in GitHub as runnable Node.js files.

[Package \(npm\)](#) | [Samples](#) | [API reference](#) | [Library source code](#) | [Give Feedback](#)

## Best practices for user delegation SAS tokens

Because anyone with the SAS token can use it to access the container and blobs, you should define the SAS token with the most restrictive permissions that still allow the token to complete the required tasks.

[Best practices for SAS tokens](#)

## Use the DefaultAzureCredential in Azure Cloud

To authenticate to Azure, *without secrets*, set up [managed identity](#). This allows your code to use [DefaultAzureCredential](#).

To set up managed identity for the Azure cloud:

- Create a managed identity
- Set the appropriate [Storage roles](#) for the identity
- Configure your Azure environment to work with your managed identity

When these two tasks are complete, use the DefaultAzureCredential instead of a connection string or account key. This allows all your environments to use the *exact same source code* without the issue of using secrets in source code.

## Use the DefaultAzureCredential in local development

In your local development environment, your Azure identity (your personal or development account you use to sign in to [Azure portal](#)) needs to [authenticate to Azure](#) to use the same code in local and cloud runtimes.

## Container: add required dependencies to your application

Include the required dependencies to create a container SAS token.

```
const {
 DefaultAzureCredential
} = require('@azure/identity');
const {
 ContainerClient,
 BlobServiceClient,
 ContainerSASPermissions,
 generateBlobSASQueryParameters,
 SASProtocol
} = require('@azure/storage-blob');

// used for local environment variables
require('dotenv').config();
```

## Container: get environment variables

The Blob Storage account name and container name are the minimum required values to create a container SAS token:

```
// Get environment variables for DefaultAzureCredential
const accountName = process.env.AZURE_STORAGE_ACCOUNT_NAME;
const containerName = process.env.AZURE_STORAGE_BLOB_CONTAINER_NAME;
```

## Create a SAS with DefaultAzureCredential

The following conceptual steps are required to create a SAS token with DefaultAzureCredential:

- Set up DefaultAzureCredential
  - Local development - use personal identity and set roles for storage
  - Azure cloud - create managed identity
- Use DefaultAzureCredential to get the user delegation key with [UserDelegationKey](#)
- Use the user delegation key to construct the SAS token with the appropriate fields with [generateBlobSASQueryParameters](#)

## Container: create SAS token with DefaultAzureCredential

With identity configured, use the following code to create **User delegation SAS token** for an existing account and container:

```

// Server creates User Delegation SAS Token for container
async function createContainerSas() {

 // Get environment variables
 const accountName = process.env.AZURE_STORAGE_ACCOUNT_NAME;
 const containerName = process.env.AZURE_STORAGE_BLOB_CONTAINER_NAME;

 // Best practice: create time limits
 const TEN_MINUTES = 10 * 60 * 1000;
 const NOW = new Date();

 // Best practice: set start time a little before current time to
 // make sure any clock issues are avoided
 const TEN_MINUTES_BEFORE_NOW = new Date(NOW.valueOf() - TEN_MINUTES);
 const TEN_MINUTES_AFTER_NOW = new Date(NOW.valueOf() + TEN_MINUTES);

 // Best practice: use managed identity - DefaultAzureCredential
 const blobServiceClient = new BlobServiceClient(
 `https://${accountName}.blob.core.windows.net`,
 new DefaultAzureCredential()
);

 // Best practice: delegation key is time-limited
 // When using a user delegation key, container must already exist
 const userDelegationKey = await blobServiceClient.getUserDelegationKey(
 TEN_MINUTES_BEFORE_NOW,
 TEN_MINUTES_AFTER_NOW
);

 // Need only list permission to list blobs
 const containerPermissionsForAnonymousUser = "l";

 // Best practice: SAS options are time-limited
 const sasOptions = {
 containerName,
 permissions: ContainerSASPermissions.parse(containerPermissionsForAnonymousUser),
 protocol: SASProtocol.HttpsAndHttp,
 startsOn: TEN_MINUTES_BEFORE_NOW,
 expiresOn: TEN_MINUTES_AFTER_NOW
 };

 const sasToken = generateBlobSASQueryParameters(
 sasOptions,
 userDelegationKey,
 accountName
).toString();

 return sasToken;
}

```

The preceding server code creates a flow of values in order to create the container SAS token:

- Create the [BlobServiceClient](#) with the [DefaultAzureCredential](#)
- Use the [blobServiceClient.getUserDelegationKey](#) operation to create a [UserDelegationKey](#)
- Use the key to create the [SAS token](#) string with [generateBlobSASQueryParameters](#)

Once you've created the container SAS token, you can provide it to the client that will consume the token. The client can then use it to list the blobs in a container. A [client code example](#) shows how to test the SAS as a consumer.

## Container: use SAS token

Once the container SAS token is created, use the token. As an example of using the SAS token, you:

- Construct a full URL including container name and query string. The query string is the SAS token.
- Create a [ContainerClient](#) with the container URL.
- Use the client: in this example, list blobs in the container with [listBlobsFlat](#).

```
// Client or another process uses SAS token to use container
async function listBlobs(sasToken){

 // Get environment variables
 const accountName = process.env.AZURE_STORAGE_ACCOUNT_NAME;
 const containerName = process.env.AZURE_STORAGE_BLOB_CONTAINER_NAME;

 // Create Url
 // SAS token is the query string with typical `?` delimiter
 const sasUrl = `https://${accountName}.blob.core.windows.net/${containerName}?${sasToken}`;
 console.log(`\nContainerUrl = ${sasUrl}\n`);

 // Create container client from SAS token url
 const containerClient = new ContainerClient(sasUrl);

 let i = 1;

 // List blobs in container
 for await (const blob of containerClient.listBlobsFlat()) {
 console.log(`Blob ${i++}: ${blob.name}`);
 }
}
```

## Blob: add required dependencies to your application

Include the required dependencies to create n blob SAS token.

```
const {
 DefaultAzureCredential
} = require('@azure/identity');
const {
 BlockBlobClient,
 BlobServiceClient,
 BlobSASPermissions,
 generateBlobSASQueryParameters,
 SASProtocol
} = require('@azure/storage-blob');

// used for local environment variables
require('dotenv').config();
```

## Blob: get environment variables

The Blob Storage account name and container name are the minimum required values to create a blob SAS token:

```
const accountName = process.env.AZURE_STORAGE_ACCOUNT_NAME;
const containerName = process.env.AZURE_STORAGE_BLOB_CONTAINER_NAME;
```

When you need to create a blob SAS token, you need to have the blob name to create the SAS token. That will be predetermined such as a random blob name, a user-submitted blob name, or a name generated from your application.

```
// Create random blob name for text file
const blobName = `${(0|Math.random()*9e6).toString(36)}.txt`;
```

## Blob: create SAS token with DefaultAzureCredential

With identity configured, use the following code to create **User delegation SAS token** for an existing account and container:

```
// Server creates User Delegation SAS Token for blob
async function createBlobSas(blobName) {

 // Get environment variables
 const accountName = process.env.AZURE_STORAGE_ACCOUNT_NAME;
 const containerName = process.env.AZURE_STORAGE_BLOB_CONTAINER_NAME;

 // Best practice: create time limits
 const TEN_MINUTES = 10 * 60 * 1000;
 const NOW = new Date();

 // Best practice: set start time a little before current time to
 // make sure any clock issues are avoided
 const TEN_MINUTES_BEFORE_NOW = new Date(NOW.valueOf() - TEN_MINUTES);
 const TEN_MINUTES_AFTER_NOW = new Date(NOW.valueOf() + TEN_MINUTES);

 // Best practice: use managed identity - DefaultAzureCredential
 const blobServiceClient = new BlobServiceClient(
 `https://${accountName}.blob.core.windows.net`,
 new DefaultAzureCredential()
);

 // Best practice: delegation key is time-limited
 // When using a user delegation key, container must already exist
 const userDelegationKey = await blobServiceClient.getUserDelegationKey(
 TEN_MINUTES_BEFORE_NOW,
 TEN_MINUTES_AFTER_NOW
);

 // Need only create/write permission to upload file
 const blobPermissionsForAnonymousUser = "cw"

 // Best practice: SAS options are time-limited
 const sasOptions = {
 blobName,
 containerName,
 permissions: BlobSASPermissions.parse(blobPermissionsForAnonymousUser),
 protocol: SASProtocol.HttpsAndHttp,
 startsOn: TEN_MINUTES_BEFORE_NOW,
 expiresOn: TEN_MINUTES_AFTER_NOW
 };

 const sasToken = generateBlobSASQueryParameters(
 sasOptions,
 userDelegationKey,
 accountName
).toString();

 return sasToken;
}
```

The preceding code creates a flow of values in order to create the container SAS token:

- Create the [BlobServiceClient](#) with [DefaultAzureCredential](#)
- Use the [blobServiceClient.getUserDelegationKey](#) operation to create a [UserDelegationKey](#)

- Use the key to create the [SAS token](#) string. If the blob name wasn't specified in the options, the SAS token is a container token.

Once you're created the blob SAS token, you can provide it to the client that will consume the token. The client can then use it to upload a blob. A [client code example](#) shows how to test the SAS as a consumer.

## Blob: use SAS token

Once the blob SAS token is created, use the token. As an example of using the SAS token, you:

- Construct a full URL including container name, blob name and query string. The query string is the SAS token.
- Create a [BlockBlobClient](#) with the container URL.
- Use the client: in this example, upload blob with [upload](#).

```
// Client or another process uses SAS token to upload content to blob
async function uploadStringToBlob(blobName, sasToken, textAsString){

 // Get environment variables
 const accountName = process.env.AZURE_STORAGE_ACCOUNT_NAME;
 const containerName = process.env.AZURE_STORAGE_BLOB_CONTAINER_NAME;

 // Create Url SAS token as query string with typical `?` delimiter
 const sasUrl = `https://${accountName}.blob.core.windows.net/${containerName}/${blobName}?${sasToken}`;
 console.log(`\nBlobUrl = ${sasUrl}\n`);

 // Create blob client from SAS token url
 const blockBlobClient = new BlockBlobClient(sasUrl);

 // Upload string
 await blockBlobClient.upload(textAsString, textAsString.length, undefined);
}
```

## See also

- [Types of SAS tokens](#)
- [API reference](#)
- [Library source code](#)
- [Give Feedback](#)

# Create a container in Azure Storage with JavaScript

8/22/2022 • 2 minutes to read • [Edit Online](#)

Blobs in Azure Storage are organized into containers. Before you can upload a blob, you must first create a container. This article shows how to create containers with the [Azure Storage client library for JavaScript](#).

The [sample code snippets](#) are available in GitHub as runnable Node.js files.

## Name a container

A container name must be a valid DNS name, as it forms part of the unique URI used to address the container or its blobs. Follow these rules when naming a container:

- Container names can be between 3 and 63 characters long.
- Container names must start with a letter or number, and can contain only lowercase letters, numbers, and the dash (-) character.
- Two or more consecutive dash characters aren't permitted in container names.

The URI for a container is in this format:

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

## Create a container

To create a container, call the following method from the `BlobStorageClient`:

- [BlobServiceClient.createContainer](#)

Containers are created immediately beneath the storage account. It's not possible to nest one container beneath another. An exception is thrown if a container with the same name already exists.

The following example creates a container asynchronously:

```
async function createContainer(blobServiceClient, containerName){

 // public access at container level
 const options = {
 access: 'container'
 };

 // creating client also creates container
 const containerClient = await blobServiceClient.createContainer(containerName, options);
 console.log(`container ${containerName} created`);

 // do something with container
 // ...

 return containerClient;
}
```

## Understand the root container

A root container, with the specific name `$root`, enables you to reference a blob at the top level of the storage account hierarchy. For example, you can reference a blob *without using a container name in the URL*.

<https://myaccount.blob.core.windowsJavaScript/default.html>

The root container must be explicitly created or deleted. It isn't created by default as part of service creation. The same code displayed in the previous section can create the root. The container name is `$root`.

## See also

- [Get started with Azure Blob Storage and JavaScript](#)
- [Create Container operation](#)
- [Delete Container operation](#)

# Delete and restore a container in Azure Storage with JavaScript

8/22/2022 • 2 minutes to read • [Edit Online](#)

This article shows how to delete containers with the [Azure Storage client library for JavaScript](#). If you've enabled container soft delete, you can restore deleted containers.

The [sample code snippets](#) are available in GitHub as runnable Node.js files.

## Delete a container

To delete a container in JavaScript, use one of the following methods:

- `BlobServiceClient.deleteContainer`
- `ContainerClient.delete`
- `ContainerClient.deleteIfExists`

After you delete a container, you can't create a container with the same name for at *least* 30 seconds. Attempting to create a container with the same name will fail with HTTP error code 409 (Conflict). Any other operations on the container or the blobs it contains will fail with HTTP error code 404 (Not Found).

## Delete container with BlobServiceClient

The following example deletes the specified container. Use the `BlobServiceClient` for the container:

```
// delete container immediately on blobServiceClient
async function deleteContainerImmediately(blobServiceClient, containerName) {
 const response = await blobServiceClient.deleteContainer(containerName);

 if (!response.errorCode) {
 console.log(`deleted ${containerItem.name} container`);
 }
}
```

## Delete container with ContainerClient

The following example shows how to delete all of the containers whose name starts with a specified prefix.

```
async function deleteContainersWithPrefix(blobServiceClient, blobNamePrefix){

 const containerOptions = {
 includeDeleted: false,
 includeMetadata: false,
 includeSystem: true,
 prefix: blobNamePrefix
 }

 for await (const containerItem of blobServiceClient.listContainers(containerOptions)) {

 const containerClient = blobServiceClient.getContainerClient(containerItem.name);

 const response = await containerClient.delete();

 if(!response.errorCode){
 console.log(`deleted ${containerItem.name} container`);
 }
 }
}
```

## Restore a deleted container

When container soft delete is enabled for a storage account, a container and its contents may be recovered after it has been deleted, within a retention period that you specify. You can restore a soft deleted container by calling.

- BlobServiceClient.[undeleteContainer](#)

The following example finds a deleted container, gets the version ID of that deleted container, and then passes that ID into the [undeleteContainer](#) method to restore the container.

```
// Undelete specific container - last version
async function undeleteContainer(blobServiceClient, containerName) {

 // version to undelete
 let containerVersion;

 const containerOptions = {
 includeDeleted: true,
 prefix: containerName
 }

 // container listing returns version (timestamp) in the ContainerItem
 for await (const containerItem of blobServiceClient.listContainers(containerOptions)) {

 // if there are multiple deleted versions of the same container,
 // the versions are in asc time order
 // the last version is the most recent
 if (containerItem.name === containerName) {
 containerVersion = containerItem.version;
 }
 }

 const containerClient = await blobServiceClient.undeleteContainer(
 containerName,
 containerVersion,

 // optional/new container name - if unused, original container name is used
 //newContainerName
);

 // undelete was successful
 console.log(`#${containerName} is undeleted`);

 // do something with containerClient
 // ...
}
}
```

## See also

- [Get started with Azure Blob Storage and JavaScript](#)
- [Soft delete for containers](#)
- [Enable and manage soft delete for containers](#)
- [Restore Container](#)

# List blob containers with JavaScript

8/22/2022 • 3 minutes to read • [Edit Online](#)

When you list the containers in an Azure Storage account from your code, you can specify a number of options to manage how results are returned from Azure Storage. This article shows how to list containers using the [Azure Storage client library for JavaScript](#).

The [sample code snippets](#) are available in GitHub as runnable Node.js files.

## Understand container listing options

To list containers in your storage account, call the following method:

- `BlobServiceClient.listContainers`

### List containers with optional prefix

By default, a listing operation returns up to 5000 results at a time.

The `BlobServiceClient.listContainers` returns a list of [ContainerItem](#) objects. Use the `containerItem.name` to create a [ContainerClient](#) in order to get a more complete [ContainerProperties](#) object.

```
async function listContainers(blobServiceClient, containerNamePrefix) {

 const options = {
 includeDeleted: false,
 includeMetadata: true,
 includeSystem: true,
 prefix: containerNamePrefix
 }

 for await (const containerItem of blobServiceClient.listContainers(options)) {

 // ContainerItem
 console.log(`For-await list: ${containerItem.name}`);

 // ContainerClient
 const containerClient = blobServiceClient.getContainerClient(containerItem.name);

 // ... do something with container
 }
}
```

## List containers with paging

To return a smaller set of results, provide a nonzero value for the size of the page of results to return.

If your storage account contains more than 5000 containers, or if you have specified a page size such that the listing operation returns a subset of containers in the storage account, then Azure Storage returns a *continuation token* with the list of containers. A continuation token is an opaque value that you can use to retrieve the next set of results from Azure Storage.

In your code, check the value of the continuation token to determine whether it is empty. When the continuation token is empty, then the set of results is complete. If the continuation token is not empty, then call the listing method again, passing in the continuation token to retrieve the next set of results, until the continuation token is empty.

```

async function listContainersWithPagingMarker(blobServiceClient) {

 // add prefix to filter list
 const containerNamePrefix = '';

 // page size
 const maxPageSize = 2;

 const options = {
 includeDeleted: false,
 includeMetadata: true,
 includeSystem: true,
 prefix: containerNamePrefix
 }

 let i = 1;
 let marker;
 let iterator = blobServiceClient.listContainers(options).byPage({ maxPageSize });
 let response = (await iterator.next()).value;

 // Prints 2 container names
 if (response.containerItems) {
 for (const container of response.containerItems) {
 console.log(`IteratorPaged: Container ${i++}: ${container.name}`);
 }
 }

 // Gets next marker
 marker = response.continuationToken;

 // Passing next marker as continuationToken
 iterator = blobServiceClient.listContainers().byPage({ continuationToken: marker, maxPageSize: maxPageSize
* 2 });
 response = (await iterator.next()).value;

 // Print next 4 container names
 if (response.containerItems) {
 for (const container of response.containerItems) {
 console.log(`Container ${i++}: ${container.name}`);
 }
 }
}

```

Use the options parameter to the `listContainers` method to filter results with a prefix.

### Filter results with a prefix

To filter the list of containers, specify a string for the `prefix` property. The prefix string can include one or more characters. Azure Storage then returns only the containers whose names start with that prefix.

```
async function listContainers(blobServiceClient, containerNamePrefix) {

 const options = {
 includeDeleted: false,
 includeMetadata: true,
 includeSystem: true,

 // filter with prefix
 prefix: containerNamePrefix
 }

 for await (const containerItem of blobServiceClient.listContainers(options)) {

 // do something with containerItem

 }
}
```

## Include metadata in results

To return container metadata with the results, specify the `metadata` value for the `BlobContainerTraits` enum. Azure Storage includes metadata with each container returned, so you do not need to fetch the container metadata as a separate operation.

```
async function listContainers(blobServiceClient, containerNamePrefix) {

 const options = {
 includeDeleted: false,
 includeSystem: true,
 prefix: containerNamePrefix,

 // include metadata
 includeMetadata: true,
 }

 for await (const containerItem of blobServiceClient.listContainers(options)) {

 // do something with containerItem

 }
}
```

## See also

- [Get started with Azure Blob Storage and JavaScript](#)
- [List Containers](#)
- [Enumerating Blob Resources](#)

# Manage container properties and metadata with JavaScript

8/22/2022 • 2 minutes to read • [Edit Online](#)

Blob containers support system properties and user-defined metadata, in addition to the data they contain. This article shows how to manage system properties and user-defined metadata with the [Azure Storage client library for JavaScript](#).

## About properties and metadata

Type	Description
<a href="#">System properties</a>	<p>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 JavaScript maintains these properties for you.</p> <p>Examples:</p> <ul style="list-style-type: none"><li>* <code>lastModified</code></li><li>* <code>leaseStatus</code></li></ul>
<a href="#">User-defined metadata</a>	<p>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.</p> <p>Examples:</p> <div style="border: 1px solid #ccc; padding: 5px; display: inline-block;"><code>project : metrics-reporting</code> <code>manager : johnh</code></div>

Metadata name/value pairs are valid HTTP headers, and so should adhere to all restrictions governing HTTP headers. Metadata names must be valid HTTP header names and should be treated as case-insensitive. Metadata values containing non-ASCII characters should be Base64-encoded or URL-encoded.

## Retrieve container properties

To retrieve container properties, use:

- `ContainerClient.getProperties()` which returns `ContainerProperties`

The following code example fetches a container's properties and writes the property values to a console window:

```
async function getContainerProperties(containerClient) {

 // Get Properties including existing metadata
 const containerProperties = await containerClient.getProperties();
 if(!containerProperties.errorCode){
 console.log(containerProperties);
 }
}
```

## Set and retrieve metadata

You can specify metadata as one or more name-value pairs container resource. To set metadata, use:

- [ContainerClient.setMetadata](#)

The name of your metadata must conform to the naming conventions for JavaScript identifiers. Metadata names preserve the case with which they were created, but are case-insensitive when set or read. If two or more metadata headers with the same name are submitted for a resource, Blob storage comma-separates and concatenates the two values and return HTTP response code 200 (OK).

The following code example sets metadata on a container.

```
/*
const metadata = {
 // values must be strings
 lastFileReview: currentDate.toString(),
 reviewer: `johnh`
}
*/
async function setContainerMetadata(containerClient, metadata) {

 await containerClient.setMetadata(metadata);

}
```

To retrieve metadata, [get the container properties](#) then use the returned **metadata** property.

- [ContainerClient.getProperties](#) which returns metadata inside the ContainerProperties object.

## See also

- [Get started with Azure Blob Storage and JavaScript](#)
- [Get Container Properties operation](#)
- [Set Container Metadata operation](#)
- [Get Container Metadata operation](#)

# Upload a blob to Azure Storage by using the JavaScript client library

8/22/2022 • 4 minutes to read • [Edit Online](#)

You can upload a blob, open a blob stream and write to that, or upload large blobs in blocks.

The [sample code snippets](#) are available in GitHub as runnable Node.js files.

## NOTE

The examples in this article assume that you've created a [BlobServiceClient](#) object by using the guidance in the [Get started with Azure Blob Storage and JavaScript](#) article. Blobs in Azure Storage are organized into containers. Before you can upload a blob, you must first create a container. To learn how to create a container, see [Create a container in Azure Storage with JavaScript](#).

## Upload by blob client

Use the following table to find the correct upload method based on the blob client.

CLIENT	UPLOAD METHOD
<a href="#">BlobClient</a>	The SDK needs to know the blob type you want to upload to. Because BlobClient is the base class for the other Blob clients, it does not have upload methods. It is mostly useful for operations that are common to the child blob classes. For uploading, create specific blob clients directly or get specific blob clients from ContainerClient.
<a href="#">BlockBlobClient</a>	This is the <b>most common upload client</b> : * <code>upload()</code> * <code>stageBlock()</code> and <code>commitBlockList()</code>
<a href="#">AppendBlobClient</a>	* <code>create()</code> * <code>append()</code>
<a href="#">PageBlobClient</a>	* <code>create()</code> * <code>appendPages()</code>

## Upload with BlockBlobClient by using a file path

The following example uploads a local file to blob storage with the [BlockBlobClient](#) object. The [options](#) object allows you to pass in your own metadata and [tags](#), used for indexing, at upload time:

```

// containerName: string
// blobName: string, includes file extension if provided
// localFileWithPath: fully qualified path and file name
// uploadOptions: {
// metadata: { reviewer: 'john', reviewDate: '2022-04-01' },
// tags: {project: 'xyz', owner: 'accounts-payable'}
// }
async function createBlobFromLocalPath(containerClient, blobName, localFileWithPath, uploadOptions){

 // create blob client from container client
 const blockBlobClient = await containerClient.getBlockBlobClient(blobName);

 // upload file to blob storage
 await blockBlobClient.uploadFile(localFileWithPath, uploadOptions);
 console.log(`"${blobName}" succeeded`);
}

```

## Upload with BlockBlobClient by using a Stream

The following example uploads a readable stream to blob storage with the [BlockBlobClient](#) object. Pass in the [BlockBlobUploadStream](#) [options](#) to affect the upload:

```

// containerName: string
// blobName: string, includes file extension if provided
// readableStream: Node.js Readable stream
// uploadOptions: {
// metadata: { reviewer: 'john', reviewDate: '2022-04-01' },
// tags: {project: 'xyz', owner: 'accounts-payable'},
// }
async function createBlobFromReadStream(containerClient, blobName, readableStream, uploadOptions) {

 // Create blob client from container client
 const blockBlobClient = await containerClient.getBlockBlobClient(blobName);

 // Size of every buffer allocated, also
 // the block size in the uploaded block blob.
 // Default value is 8MB
 const bufferSize = 4 * 1024 * 1024;

 // Max concurrency indicates the max number of
 // buffers that can be allocated, positive correlation
 // with max uploading concurrency. Default value is 5
 const maxConcurrency = 20;

 // use transform per chunk - only to see chunck
 const transformedReadableStream = readableStream.pipe(myTransform);

 // Upload stream
 await blockBlobClient.uploadStream(transformedReadableStream, bufferSize, maxConcurrency, uploadOptions);

 // do something with blob
 const getTagsResponse = await blockBlobClient.getTags();
 console.log(`tags for ${blobName} = ${JSON.stringify(getTagsResponse.tags)} `);
}

```

Transform the stream during the upload for data clean up.

```
// Transform stream
// Reasons to transform:
// 1. Sanitize the data - remove PII
// 2. Compress or uncompress
const myTransform = new Transform({
 transform(chunk, encoding, callback) {
 // see what is in the artificially
 // small chunk
 console.log(chunk);
 callback(null, chunk);
 },
 decodeStrings: false
});
```

The following code demonstrates how to use the function.

```
// fully qualified path to file
const localFileWithPath = path.join(__dirname, `my-text-file.txt`);

// encoding: just to see the chunk as it goes by in the transform
const streamOptions = { highWaterMark: 20, encoding: 'utf-8' }

const readableStream = fs.createReadStream(localFileWithPath, streamOptions);

// upload options
const uploadOptions = {

 // not indexed for searching
 metadata: {
 owner: 'PhillyProject'
 },

 // indexed for searching
 tags: {
 createdBy: 'YOUR-NAME',
 createdWith: `StorageSnippetsForDocs-${i}`,
 createdOn: (new Date()).toDateString()
 }
}

// upload stream
await createBlobFromReadStream(containerClient, `my-text-file.txt`, readableStream, uploadOptions);
```

## Upload with BlockBlobClient by using a BinaryData object

The following example uploads a Node.js buffer to blob storage with the [BlockBlobClient](#) object. Pass in the [BlockBlobParallelUpload](#) [options](#) to affect the upload:

```

// containerName: string
// blobName: string, includes file extension if provided
// buffer: blob content
// uploadOptions: {
// blockSize: destination block blob size in bytes,
// concurrency: concurrency of parallel uploading - must be greater than or equal to 0,
// maxSingleShotSize: blob size threshold in bytes to start concurrency uploading
// metadata: { reviewer: 'john', reviewDate: '2022-04-01' },
// tags: {project: 'xyz', owner: 'accounts-payable'}
// }
async function createBlobFromBuffer(containerClient, blobName, buffer, uploadOptions) {

 // Create blob client from container client
 const blockBlobClient = await containerClient.getBlockBlobClient(blobName);

 // Upload buffer
 await blockBlobClient.uploadData(buffer, uploadOptions);

 // do something with blob
 const getTagsResponse = await blockBlobClient.getTags();
 console.log(`tags for ${blobName} = ${JSON.stringify(getTagsResponse.tags)}`);
}

```

The following code demonstrates how to use the function.

```

// fully qualified path to file
const localFileWithPath = path.join(__dirname, `daisies.jpg`);

// read file into buffer
const buffer = await fs.readFile(localFileWithPath);

// upload options
const uploadOptions = {

 // not indexed for searching
 metadata: {
 owner: 'PhillyProject'
 },

 // indexed for searching
 tags: {
 createdBy: 'YOUR-NAME',
 createdWith: `StorageSnippetsForDocs-${i}`,
 createdOn: (new Date()).toDateString()
 }
}

// upload buffer
createBlobFromBuffer(containerClient, `daisies.jpg`, buffer, uploadOptions)

```

## Upload a string with BlockBlobClient

The following example uploads a string to blob storage with the [BlockBlobClient](#) object. Pass in the [BlockBlobUploadOptions](#) [options](#) to affect the upload:

```
// containerClient: container client
// blobName: string, includes file extension if provided
// fileContentsAsString: blob content
// uploadOptions: {
// metadata: { reviewer: 'john', reviewDate: '2022-04-01' },
// tags: {project: 'xyz', owner: 'accounts-payable'}
// }
async function createBlobFromString(containerClient, blobName, fileContentsAsString, uploadOptions){

 // Create blob client from container client
 const blockBlobClient = await containerClient.getBlockBlobClient(blobName);

 // Upload string
 await blockBlobClient.upload(fileContentsAsString, fileContentsAsString.length, uploadOptions);

 // do something with blob
 const getTagsResponse = await blockBlobClient.getTags();
 console.log(`tags for ${blobName} = ${JSON.stringify(getTagsResponse.tags)}`);
}
```

## See also

- [Manage and find Azure Blob data with blob index tags](#)
- [Use blob index tags to manage and find data on Azure Blob Storage](#)
- [Put Blob \(REST API\)](#)
- [Put Blob From URL \(REST API\)](#)

# Download a blob in Azure Storage using the JavaScript client library

8/22/2022 • 2 minutes to read • [Edit Online](#)

You can download a blob by using any of the following methods:

- [Blob.download](#)
- [Blob.downloadToBuffer](#)
- [Blob.downloadToFile](#)

The [sample code snippets](#) are available in GitHub as runnable Node.js files.

## NOTE

The examples in this article assume that you've created a [BlobServiceClient](#) object by using the guidance in the [Get started with Azure Blob Storage and JavaScript](#) article. Blobs in Azure Storage are organized into containers. Before you can upload a blob, you must first create a container. To learn how to create a container, see [Create a container in Azure Storage with JavaScript](#).

## Download to a file path

The following example downloads a blob by using a file path with the [BlobClient.downloadToFile](#) method:

```
async function downloadBlobToFile(containerClient, blobName, fileNameWithPath) {
 const blobClient = await containerClient.getBlobClient(blobName);

 await blobClient.downloadToFile(fileNameWithPath);
 console.log(`download of ${blobName} success`);
}
```

## Download as a stream

The following example downloads a blob by creating a Node.js writable stream object and then piping to that stream with the [BlobClient.download](#) method.

```
async function downloadBlobAsStream(containerClient, blobName, writableStream) {
 const blobClient = await containerClient.getBlobClient(blobName);

 const downloadResponse = await blobClient.download();

 downloadResponse.readableStreamBody.pipe(writableStream);
 console.log(`download of ${blobName} succeeded`);
}
```

## Download to a string

The following example downloads a blob to a string with [BlobClient.download](#) method.

```
async function downloadBlobToString(containerClient, blobName) {
 const blobClient = await containerClient.getBlobClient(blobName);

 const downloadResponse = await blobClient.download();

 const downloaded = await streamToBuffer(downloadResponse.readableStreamBody);
 console.log('Downloaded blob content:', downloaded.toString());
}

async function streamToBuffer(readableStream) {
 return new Promise((resolve, reject) => {
 const chunks = [];
 readableStream.on('data', (data) => {
 chunks.push(data instanceof Buffer ? data : Buffer.from(data));
 });
 readableStream.on('end', () => {
 resolve(Buffer.concat(chunks));
 });
 readableStream.on('error', reject);
 });
}
```

## See also

- [Get started with Azure Blob Storage and JavaScript](#)
- [DownloadStreaming](#)
- [Get Blob \(REST API\)](#)

# Copy a blob with Azure Storage using the JavaScript client library

8/22/2022 • 3 minutes to read • [Edit Online](#)

This article demonstrates how to copy a blob in an Azure Storage account. It also shows how to abort an asynchronous copy operation. The example code uses the Azure Storage client libraries.

The [sample code snippets](#) are available in GitHub as runnable Node.js files.

## NOTE

The examples in this article assume that you've created a [BlobServiceClient](#) object by using the guidance in the [Get started with Azure Blob Storage and JavaScript](#) article. Blobs in Azure Storage are organized into containers. Before you can upload a blob, you must first create a container. To learn how to create a container, see [Create a container in Azure Storage with JavaScript](#).

## About copying blobs

When you copy a blob within the same storage account, it's a synchronous operation. When you copy across accounts it's an asynchronous operation.

The source blob for a copy operation may be a block blob, an append blob, a page blob, or a snapshot. If the destination blob already exists, it must be of the same blob type as the source blob. An existing destination blob will be overwritten.

The destination blob can't be modified while a copy operation is in progress. A destination blob can only have one outstanding copy operation. In other words, a blob can't be the destination for multiple pending copy operations.

The entire source blob or file is always copied. Copying a range of bytes or set of blocks is not supported.

When a blob is copied, its system properties are copied to the destination blob with the same values.

A copy operation can take any of the following forms:

- Copy a source blob to a destination blob with a different name. The destination blob can be an existing blob of the same blob type (block, append, or page), or can be a new blob created by the copy operation.
- Copy a source blob to a destination blob with the same name, effectively replacing the destination blob. Such a copy operation removes any uncommitted blocks and overwrites the destination blob's metadata.
- Copy a source file in the Azure File service to a destination blob. The destination blob can be an existing block blob, or can be a new block blob created by the copy operation. Copying from files to page blobs or append blobs is not supported.
- Copy a snapshot over its base blob. By promoting a snapshot to the position of the base blob, you can restore an earlier version of a blob.
- Copy a snapshot to a destination blob with a different name. The resulting destination blob is a writeable blob and not a snapshot.

## Copy a blob

To copy a blob, use the [BlobClient.beginCopyFromURL method](#). The following code example gets a [BlobClient](#)

representing a previously created blob and copies it to a new blob:

```
async function copyBlob(
 blobServiceClient,
 sourceBlobContainerName,
 sourceBlobName,
 destinationBlobContainerName,
 destinationBlobName) {

 // create container clients
 const sourceContainerClient = blobServiceClient.getContainerClient(sourceBlobContainerName);
 const destinationContainerClient = blobServiceClient.getContainerClient(destinationBlobContainerName);

 // create blob clients
 const sourceBlobClient = await sourceContainerClient.getBlobClient(sourceBlobName);
 const destinationBlobClient = await destinationContainerClient.getBlobClient(destinationBlobName);

 // start copy
 const copyPoller = await destinationBlobClient.beginCopyFromURL(sourceBlobClient.url);
 console.log('start copy from A to B');

 // wait until done
 await copyPoller.pollUntilDone();
}
```

## Cancel a copy operation

When you abort a copy operation, the destination blob's property, [copyStatus](#), is set to [aborted](#).

```
async function copyThenAbortBlob(
 blobServiceClient,
 sourceBlobContainerName,
 sourceBlobName,
 destinationBlobContainerName,
 destinationBlobName) {

 // create container clients
 const sourceContainerClient = blobServiceClient.getContainerClient(sourceBlobContainerName);
 const destinationContainerClient = blobServiceClient.getContainerClient(destinationBlobContainerName);

 // create blob clients
 const sourceBlobClient = await sourceContainerClient.getBlobClient(sourceBlobName);
 const destinationBlobClient = await destinationContainerClient.getBlobClient(destinationBlobName);

 // start copy
 const copyPoller = await destinationBlobClient.beginCopyFromURL(sourceBlobClient.url);
 console.log('start copy from A to C');

 // cancel operation after starting it -
 // sample file may be too small to be canceled.
 try {
 await copyPoller.cancelOperation();
 console.log('request to cancel copy from A to C');

 // calls to get the result now throw PollerCancelledError
 await copyPoller.getResult();
 } catch (err) {
 if (err.name === 'PollerCancelledError') {
 console.log('The copy was cancelled.');
 }
 }
}
```

## Abort a copy operation

Aborting a copy operation, with [BlobClient.abortCopyFromURL](#) results in a destination blob of zero length. However, the metadata for the destination blob will have the new values copied from the source blob or set explicitly during the copy operation. To keep the original metadata from before the copy, make a snapshot of the destination blob before calling one of the copy methods. The final blob will be committed when the copy completes.

## See also

- [Copy Blob](#)
- [Abort Copy Blob](#)
- [Get started with Azure Blob Storage and JavaScript](#)

# List blobs using the Azure Storage client library for JavaScript

8/22/2022 • 4 minutes to read • [Edit Online](#)

When you list blobs from your code, you can specify a number of options to manage how results are returned from Azure Storage. You can specify the number of results to return in each set of results, and then retrieve the subsequent sets. You can specify a prefix to return blobs whose names begin with that character or string. And you can list blobs in a flat listing structure, or hierarchically. A hierarchical listing returns blobs as though they were organized into folders.

The [sample code snippets](#) are available in GitHub as runnable Node.js files.

## Understand blob listing options

To list the blobs in a storage account, call one of these methods:

- [ContainerClient.listBlobsByHierarchy](#)
- [ContainerClient.listBlobsFlat](#)

Related functionality can be found in the following methods:

- [BlobServiceClient.findBlobsByTag](#)
- [ContainerClient.findBlobsByTag](#)

### Manage how many results are returned

By default, a listing operation returns up to 5000 results at a time, but you can specify the number of results that you want each listing operation to return. The examples presented in this article show you how to return results in pages.

### Filter results with a prefix

To filter the list of blobs, specify a string for the `prefix` property in the [list options](#). The prefix string can include one or more characters. Azure Storage then returns only the blobs whose names start with that prefix.

```
const listOptions = {
 includeCopy: false, // include metadata from previous copies
 includeDeleted: false, // include deleted blobs
 includeDeletedWithVersions: false, // include deleted blobs with versions
 includeLegalHost: false, // include legal host id
 includeMetadata: true, // include custom metadata
 includeSnapshots: true, // include snapshots
 includeTags: true, // include indexable tags
 includeUncommittedBlobs: false, // include uncommitted blobs
 includeVersions: false, // include all blob version
 prefix: '' // filter by blob name prefix
};
```

### Return metadata

You can return blob metadata with the results by specifying the `includeMetadata` property in the [list options](#).

### Flat listing versus hierarchical listing

Blobs in Azure Storage are organized in a flat paradigm, rather than a hierarchical paradigm (like a classic file system). However, you can organize blobs into *virtual directories* in order to mimic a folder structure. A virtual

directory forms part of the name of the blob and is indicated by the delimiter character.

To organize blobs into virtual directories, use a delimiter character in the blob name. The default delimiter character is a forward slash (/), but you can specify any character as the delimiter.

If you name your blobs using a delimiter, then you can choose to list blobs hierarchically. For a hierarchical listing operation, Azure Storage returns any virtual directories and blobs beneath the parent object. You can call the listing operation recursively to traverse the hierarchy, similar to how you would traverse a classic file system programmatically.

If you've enabled the hierarchical namespace feature on your account, directories are not virtual. Instead, they are concrete, independent objects. Therefore, directories appear in the list as zero-length blobs.

## Use a flat listing

By default, a listing operation returns blobs in a flat listing. In a flat listing, blobs are not organized by virtual directory.

The following example lists the blobs in the specified container using a flat listing.

```
async function listBlobsFlatWithPageMarker(containerClient) {

 // page size - artificially low as example
 const maxPageSize = 2;

 let i = 1;
 let marker;

 // some options for filtering list
 const listOptions = {
 includeMetadata: true,
 includeSnapshots: false,
 includeTags: true,
 includeVersions: false,
 prefix: ''
 };

 let iterator = containerClient.listBlobsFlat(listOptions).byPage({ maxPageSize });
 let response = (await iterator.next()).value;

 // Prints blob names
 for (const blob of response.segment.blobItems) {
 console.log(`Flat listing: ${i++}: ${blob.name}`);
 }

 // Gets next marker
 marker = response.continuationToken;

 // Passing next marker as continuationToken
 iterator = containerClient.listBlobsFlat().byPage({
 continuationToken: marker,
 maxPageSize: maxPageSize * 2
 });
 response = (await iterator.next()).value;

 // Prints next blob names
 for (const blob of response.segment.blobItems) {
 console.log(`Flat listing: ${i++}: ${blob.name}`);
 }
}
```

The sample output is similar to:

```
Flat listing: 1: a0/blob-0.txt
Flat listing: 2: a1/blob-1.txt
Flat listing: 3: a2/blob-2.txt
```

## Use a hierarchical listing

When you call a listing operation hierarchically, Azure Storage returns the virtual directories and blobs at the first level of the hierarchy.

To list blobs hierarchically, call the [BlobContainerClient.listBlobsByHierarchy](#) method.

The following example lists the blobs in the specified container using a hierarchical listing, with an optional segment size specified, and writes the blob name to the console window.

```
// Recursively list virtual folders and blobs
async function listBlobHierarchical(containerClient, virtualHierarchyDelimiter='/') {

 // page size - artificially low as example
 const maxPageSize = 2;

 // some options for filtering list
 const listOptions = {
 includeMetadata: true,
 includeSnapshots: false,
 includeTags: true,
 includeVersions: false,
 prefix: ''
 };

 let i = 1;
 console.log(`Folder ${virtualHierarchyDelimiter}`);

 for await (const response of containerClient
 .listBlobsByHierarchy(virtualHierarchyDelimiter, listOptions)
 .byPage({ maxPageSize })) {

 console.log(` Page ${i++}`);
 const segment = response.segment;

 if (segment.blobPrefixes) {

 // Do something with each virtual folder
 for await (const prefix of segment.blobPrefixes) {

 // build new virtualHierarchyDelimiter from current and next
 await listBlobHierarchical(containerClient, `${virtualHierarchyDelimiter}${prefix.name}`);
 }
 }

 for (const blob of response.segment.blobItems) {

 // Do something with each blob
 console.log(`\tBlobItem: name - ${blob.name}`);
 }
 }
}
```

The sample output is similar to:

```
Hier listing: Folder /
 Page 1
Hier listing: Folder /a0/
 Page 1
 BlobItem: name - a0/blob-0.txt
 BlobItem: name - a1/blob-1.txt
 Page 2
 BlobItem: name - a2/blob-2.txt
Hier listing: Folder /a1/
 Page 1
 BlobItem: name - a0/blob-0.txt
 BlobItem: name - a1/blob-1.txt
 Page 2
 BlobItem: name - a2/blob-2.txt
 Page 2
Hier listing: Folder /a2/
 Page 1
 BlobItem: name - a0/blob-0.txt
 BlobItem: name - a1/blob-1.txt
 Page 2
 BlobItem: name - a2/blob-2.txt
```

#### NOTE

Blob snapshots cannot be listed in a hierarchical listing operation.

## Next steps

- [List Blobs](#)
- [Enumerating Blob Resources](#)
- [Blob versioning](#)

# Delete and restore a blob in your Azure Storage account using the JavaScript client library

8/22/2022 • 2 minutes to read • [Edit Online](#)

This article shows how to delete blobs with the [Azure Storage client library for JavaScript](#). If you've enabled blob soft delete, you can restore deleted blobs.

The [sample code snippets](#) are available in GitHub as runnable Node.js files.

## NOTE

The examples in this article assume that you've created a [BlobServiceClient](#) object by using the guidance in the [Get started with Azure Blob Storage and JavaScript](#) article. Blobs in Azure Storage are organized into containers. Before you can upload a blob, you must first create a container. To learn how to create a container, see [Create a container in Azure Storage with JavaScript](#).

## Delete a blob

To delete a blob, call either of these methods:

- [BlobClient.delete](#)
- [BlobClient.deleteIfExists](#)

The following example deletes a blob.

```
async function deleteBlob(containerClient, blobName){

 // include: Delete the base blob and all of its snapshots.
 // only: Delete only the blob's snapshots and not the blob itself.
 const options = {
 deleteSnapshots: 'include' // or 'only'
 }

 // Create blob client from container client
 const blockBlobClient = await containerClient.getBlockBlobClient(blobName);

 await blockBlobClient.delete(options);

 console.log(`deleted blob ${blobName}`);
}
```

The following example deletes a blob if it exists.

```
async function deleteBlobIfExists(containerClient, blobName){

 // include: Delete the base blob and all of its snapshots.
 // only: Delete only the blob's snapshots and not the blob itself.
 const options = {
 deleteS snapshots: 'include' // or 'only'
 }

 // Create blob client from container client
 const blockBlobClient = await containerClient.getBlockBlobClient(blobName);

 await blockBlobClient.deleteIfExists(options);

 console.log(`deleted blob ${blobName}`);
}
```

## Restore a deleted blob

Blob soft delete protects an individual blob and its versions, snapshots, and metadata 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 the blob to its state at deletion. After the retention period has expired, the blob is permanently deleted. For more information about blob soft delete, see [Soft delete for blobs](#).

You can use the Azure Storage client libraries to restore a soft-deleted blob or snapshot.

### Restore soft-deleted objects when versioning is disabled

To restore deleted blobs, call the following method:

- [ContainerClient.undelete](#)

This method restores soft-deleted blobs and any deleted snapshots associated with it. Calling this method for a blob that has not been deleted has no effect.

```
async function undeleteBlob(containerClient, blobName){

 // Create blob client from container client
 const blockBlobClient = await containerClient.getBlockBlobClient(blobName);

 await blockBlobClient.undelete();

 console.log(`undeleted blob ${blobName}`);
}
```

## See also

- [Get started with Azure Blob Storage and JavaScript](#)
- [Delete Blob \(REST API\)](#)
- [Soft delete for blobs](#)
- [Undelete Blob \(REST API\)](#)

# Use blob index tags to manage and find data in Azure Blob Storage (JavaScript)

8/22/2022 • 3 minutes to read • [Edit Online](#)

Blob index tags categorize data in your storage account using key-value tag attributes. These tags are automatically indexed and exposed as a searchable multi-dimensional index to easily find data. This article shows you how to set, get, and find data using blob index tags.

To learn more about this feature along with known issues and limitations, see [Manage and find Azure Blob data with blob index tags](#).

The [sample code snippets](#) are available in GitHub as runnable Node.js files.

## NOTE

The examples in this article assume that you've created a [BlobServiceClient](#) object by using the guidance in the [Get started with Azure Blob Storage and JavaScript](#) article. Blobs in Azure Storage are organized into containers. Before you can upload a blob, you must first create a container. To learn how to create a container, see [Create a container in Azure Storage with JavaScript](#).

## Set and retrieve index tags

You can set and get index tags if your code has authorized access by using an account key or if your code uses a security principal that has been given the appropriate role assignments. For more information, see [Manage and find Azure Blob data with blob index tags](#).

### Set tags

You can set tags at blob upload time or by using the following method:

- [BlobClient.setTags](#)

The following example performs this task.

```
// A blob can have up to 10 tags.
//
// const tags = {
// project: 'End of month billing summary',
// reportOwner: 'John Doe',
// reportPresented: 'April 2022'
// }
async function setTags(containerClient, blobName, tags) {

 // Create blob client from container client
 const blockBlobClient = await containerClient.getBlockBlobClient(blobName);

 // Set tags
 await blockBlobClient.setTags(tags);

 console.log(`uploading blob ${blobName}`);
}
```

You can delete all tags by passing an empty JSON object into the `setTags` method.

## RELATED ARTICLES

[Manage and find Azure Blob data with blob index tags](#)

[Set Blob Tags \(REST API\)](#)

### Get tags

You can get tags by using either of the following methods:

- [BlobClient.getTags](#)

The following example shows how to get and iterate over the blob's tags.

```
async function getTags(containerClient, blobName) {

 // Create blob client from container client
 const blockBlobClient = await containerClient.getBlockBlobClient(blobName);

 // Get tags
 const result = await blockBlobClient.getTags();

 for (const tag in result.tags) {

 console.log(`TAG: ${tag}: ${result.tags[tag]}`);
 }
}
```

## Filter and find data with blob index tags

You can use index tags to find and filter data if your code has authorized access by using an account key or if your code uses a security principal that has been given the appropriate role assignments. For more information, see [Manage and find Azure Blob data with blob index tags](#).

### NOTE

You can't use index tags to retrieve previous versions. Tags for previous versions aren't passed to the blob index engine. For more information, see [Conditions and known issues](#).

Data is queried with a JSON object sent as a string. The properties don't need to have additional string quotes but the values do need additional string quotes.

The following table shows some query strings:

QUERY STRING FOR TAGS (TAGODATAQUERY)	DESCRIPTION
<code>id='1' AND project='billing'</code>	Filter blobs across all containers based on these two properties
<code>owner='PhillyProject' AND createdOn &gt;= '2021-12' AND createdOn &lt;= '2022-06'</code>	Filter blobs across all containers based on strict property value for <code>owner</code> and range of dates for <code>createdOn</code> property.
<code>@container = 'my-container' AND createdBy = 'Jill'</code>	Filter by <code>container</code> and specific property. In this query, <code>createdBy</code> is a text match and doesn't indicate an authorization match through Active Directory.

You can find data by using the following method:

- [BlobServiceClient.findBlobsByTags](#)

The following example finds all blobs matching the tagOdataQuery parameter.

```
async function findBlobsByQuery(blobServiceClient, tagOdataQuery) {

 // page size
 const maxPageSize = 10;

 let i = 1;
 let marker;

 const listOptions = {
 includeMetadata: true,
 includeSnapshots: false,
 includeTags: true,
 includeVersions: false
 };

 let iterator = blobServiceClient.findBlobsByTags(tagOdataQuery, listOptions).byPage({ maxPageSize });
 let response = (await iterator.next()).value;

 // Prints blob names
 if (response.blobs) {
 for (const blob of response.blobs) {
 console.log(`Blob ${i++}: ${blob.name} - ${JSON.stringify(blob.tags)})`);
 }
 }

 // Gets next marker
 marker = response.continuationToken;

 // no more blobs
 if (!marker) return;

 // Passing next marker as continuationToken
 iterator = blobServiceClient
 .findBlobsByTags(tagOdataQuery, listOptions)
 .byPage({ continuationToken: marker, maxPageSize });
 response = (await iterator.next()).value;

 // Prints blob names
 if (response.blobs) {
 for (const blob of response.blobs) {
 console.log(`Blob ${i++}: ${blob.name} - ${JSON.stringify(blob.tags)})`);
 }
 }
}
```

And example output for this function shows the matched blobs and their tags, based on the `console.log` code in the preceding function:

#### RESPONSE

```
Blob 1: set-tags-1650565920363-query-by-tag-blob-a-1.txt - {"createdOn":"2022-01","owner":"PhillyProject","project":"set-tags-1650565920363"}
```

## See also

- [Manage and find Azure Blob data with blob index tags](#)
- [Get Blob Tags \(REST API\)](#)

- [Find Blobs by Tags](#) (REST API)

# Manage blob properties and metadata with JavaScript

8/22/2022 • 3 minutes to read • [Edit Online](#)

In addition to the data they contain, blobs support system properties and user-defined metadata. This article shows how to manage system properties and user-defined metadata with the [Azure Storage client library for JavaScript](#).

The [sample code snippets](#) are available in GitHub as runnable Node.js files.

## 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 JavaScript 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.

### NOTE

Blob index tags also provide the ability to store arbitrary user-defined key/value attributes alongside an Azure Blob storage resource. While similar to metadata, only blob index tags are automatically indexed and made searchable by the native blob service. Metadata cannot be indexed and queried unless you utilize a separate service such as Azure Search.

To learn more about this feature, see [Manage and find data on Azure Blob storage with blob index \(preview\)](#).

## Set blob http headers

The following code example sets blob HTTP system properties on a blob.

To set the HTTP properties for a blob, call [BlobClient.setHTTPHeaders](#). Review the [BlobHTTPHeaders properties](#) to know which HTTP properties you want to set. Any HTTP properties not explicitly set are cleared.

```
/*
properties= {
 blobContentType: 'text/plain',
 blobContentLanguage: 'en-us',
 blobContentEncoding: 'utf-8',
 // all other http properties are cleared
}
*/
async function setHTTPHeaders(blobClient, headers) {

 await blobClient.setHTTPHeaders(headers);

 console.log(`headers set successfully`);
}
```

## Set metadata

You can specify metadata as one or more name-value pairs on a blob or container resource. To set metadata, send a JSON object of name-value pairs with

- [BlobClient.setMetadata](#) returns a [BlobGetPropertiesResponse](#) object.

Metadata name/value pairs are valid HTTP headers and should adhere to all restrictions governing HTTP headers. Metadata names must be valid HTTP header names and valid C# identifiers, may contain only ASCII characters, and should be treated as case-insensitive. Either [Base64-encode](#) or [URL-encode](#) your metadata values containing non-ASCII characters.

Metadata names maintain the case used when they were created, but are case-insensitive when set or read. If two or more metadata headers using the same name are submitted for a resource, Azure Blob storage returns HTTP error code 400 (Bad Request).

The following code example sets metadata on a blob.

```
/*
metadata= {
 reviewedBy: 'Bob',
 releasedBy: 'Jill',
}
*/
async function setBlobMetadata(blobClient, metadata) {

 await blobClient.setMetadata(metadata);

 console.log(`metadata set successfully`);

}
```

To read the metadata, get the blob's properties (shown below), specifically referencing the `metadata` property.

## Get blob properties

The following code example gets a blob's system properties, including HTTP headers and metadata, and displays those values.

```
async function getProperties(blobClient) {

 const properties = await blobClient.getProperties();
 console.log(blobClient.name + ' properties: ');

 for (const property in properties) {

 switch (property) {
 // nested properties are stringified and returned as strings
 case 'metadata':
 case 'objectReplicationRules':
 console.log(` ${property}: ${JSON.stringify(properties[property])}`);
 break;
 default:
 console.log(` ${property}: ${properties[property]}`);
 break;
 }
 }
}
```

The output for these `console.log` lines looks like:

```
my-blob.txt properties:
 lastModified: Thu Apr 21 2022 13:02:53 GMT-0700 (Pacific Daylight Time)
 createdOn: Thu Apr 21 2022 13:02:53 GMT-0700 (Pacific Daylight Time)
 metadata: {"releasedby":"Jill","reviewedby":"Bob"}
 objectReplicationPolicyId: undefined
 objectReplicationRules: {}
 blobType: BlockBlob
 copyCompletedOn: undefined
 copyStatusDescription: undefined
 copyId: undefined
 copyProgress: undefined
 copySource: undefined
 copyStatus: undefined
 isIncrementalCopy: undefined
 destinationSnapshot: undefined
 leaseDuration: undefined
 leaseState: available
 leaseStatus: unlocked
 contentLength: 19
 contentType: text/plain
 etag: "0x8DA23D1EBA8E607"
 contentMD5: undefined
 contentEncoding: utf-8
 contentDisposition: undefined
 contentLanguage: en-us
 cacheControl: undefined
 blobSequenceNumber: undefined
 clientRequestId: 58da0441-7224-4837-9b4a-547f9a0c7143
 requestId: 26acb38a-001e-0046-27ba-55ef22000000
 version: 2021-04-10
 date: Thu Apr 21 2022 13:02:52 GMT-0700 (Pacific Daylight Time)
 acceptRanges: bytes
 blobCommittedBlockCount: undefined
 isServerEncrypted: true
 encryptionKeySha256: undefined
 encryptionScope: undefined
 accessTier: Hot
 accessTierInferred: true
 archiveStatus: undefined
 accessTierChangedOn: undefined
 versionId: undefined
 isCurrentVersion: undefined
 tagCount: undefined
 expiresOn: undefined
 isSealed: undefined
 rehydratePriority: undefined
 lastAccessed: undefined
 immutabilityPolicyExpiresOn: undefined
 immutabilityPolicyMode: undefined
 legalHold: undefined
 errorCode: undefined
 body: true
 _response: [object Object]
 objectReplicationDestinationPolicyId: undefined
 objectReplicationSourceProperties:
```

## See also

- [Set Blob Properties operation](#)
- [Get Blob Properties operation](#)
- [Set Blob Metadata operation](#)
- [Get Blob Metadata operation](#)

# Use the Azurite emulator for local Azure Storage development

8/22/2022 • 17 minutes to read • [Edit Online](#)

The Azurite open-source emulator provides a free local environment for testing your Azure blob, queue storage, and table storage applications. When you're satisfied with how your application is working locally, switch to using an Azure Storage account in the cloud. The emulator provides cross-platform support on Windows, Linux, and macOS.

Azurite is the future storage emulator platform. Azurite supersedes the [Azure Storage Emulator](#). Azurite will continue to be updated to support the latest versions of Azure Storage APIs.

There are several different ways to install and run Azurite on your local system. Select any of these tabs.

## Install Azurite

- [Visual Studio](#)
- [Visual Studio Code](#)
- [npm](#)
- [Docker Hub](#)
- [GitHub](#)

Azurite is automatically available with [Visual Studio 2022](#). If you are running an earlier version of Visual Studio, you'll need to install Azurite by using either Node Package Manager, DockerHub, or by cloning the Azurite GitHub repository.

## Run Azurite

- [Visual Studio](#)
- [Visual Studio Code](#)
- [npm](#)
- [Docker Hub](#)
- [GitHub](#)

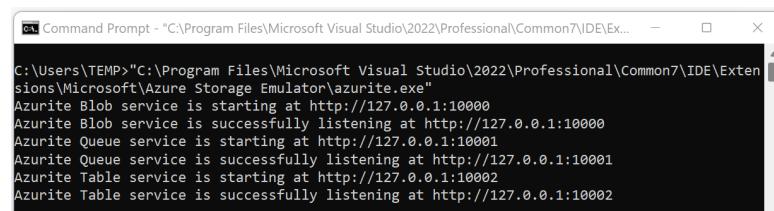
With a few configurations, Azure Functions or ASP.NET projects start Azurite automatically. For all other project types, you'll have to start Azurite from the command line.

### Running Azurite from the command line

You can find the Azurite executable file in the extensions folder of your Visual Studio installation. The specific location can vary based on which version of Visual Studio you have installed. For example, if you've installed Visual Studio 2022 professional edition on a Windows computer or Virtual Machine (VM), you would find the Azurite executable file at this location:

```
C:\Program Files\Microsoft Visual Studio\2022\Professional\Common7\IDE\Extensions\Microsoft\Azure Storage Emulator
```

After you run the executable file, Azurite listens for connections.

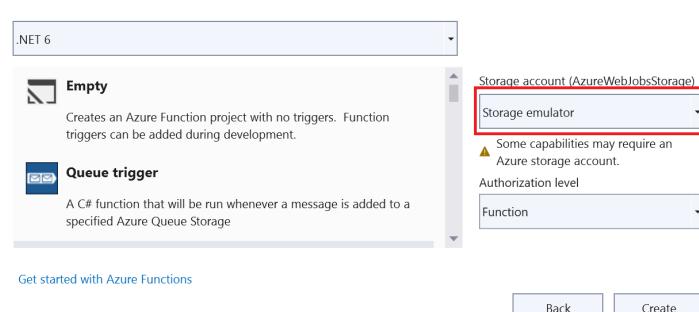


```
C:\Users\TEMP>"C:\Program Files\Microsoft Visual Studio\2022\Professional\Common7\IDE\Extensions\Microsoft\Azure Storage Emulator\azurite.exe"
Azurite Blob service is starting at http://127.0.0.1:10000
Azurite Blob service is successfully listening at http://127.0.0.1:10000
Azurite Queue service is starting at http://127.0.0.1:10001
Azurite Queue service is successfully listening at http://127.0.0.1:10001
Azurite Table service is starting at http://127.0.0.1:10002
Azurite Table service is successfully listening at http://127.0.0.1:10002
```

### Running Azurite from an Azure Functions project

In Visual Studio 2022, create an [Azure Functions](#) project. As you create the project, choose the **Storage Emulator**.

## Create a new Azure Functions application



After you create the project, Azurite starts automatically.

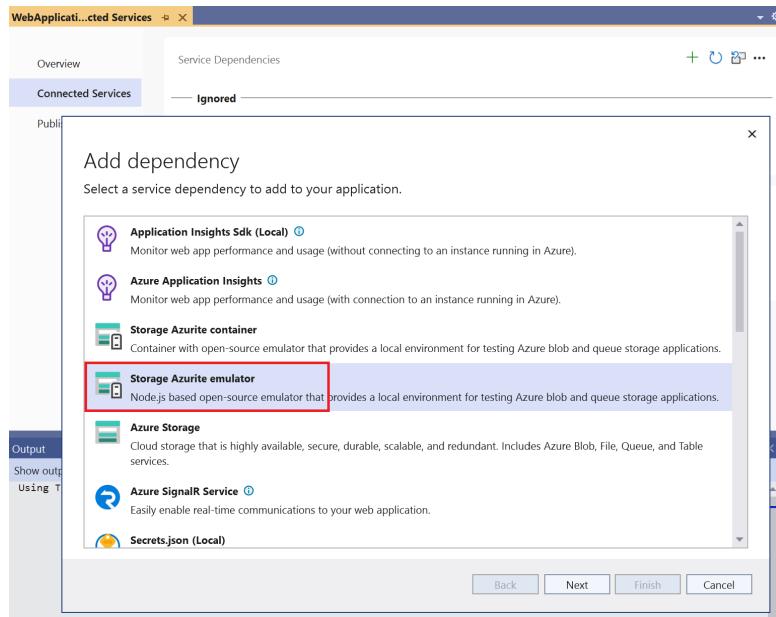
```

Output
Show output from: Service Dependencies
Ensuring Azure Functions Core Tools are up to date. This may take a few minutes...
Azure Functions Core Tools are up to date.
FunctionApp6: c:\program files\microsoft visual studio\2022\professional\common7\ide\extensions\microsoft\Azure Storage
Ensuring Azure Functions Core Tools are up to date. This may take a few minutes...
Azure Functions Core Tools are up to date.
FunctionApp6: c:\program files\microsoft visual studio\2022\professional\common7\ide\extensions\microsoft\Azure Storage
FunctionApp6: Azurite Blob service is starting at http://127.0.0.1:10000
FunctionApp6: Azurite Blob service is successfully listening at http://127.0.0.1:10000
FunctionApp6: Azurite Queue service is starting at http://127.0.0.1:10001
FunctionApp6: Azurite Queue service is successfully listening at http://127.0.0.1:10001
FunctionApp6: Azurite Table service is starting at http://127.0.0.1:10002
FunctionApp6: Azurite Table service is successfully listening at http://127.0.0.1:10002

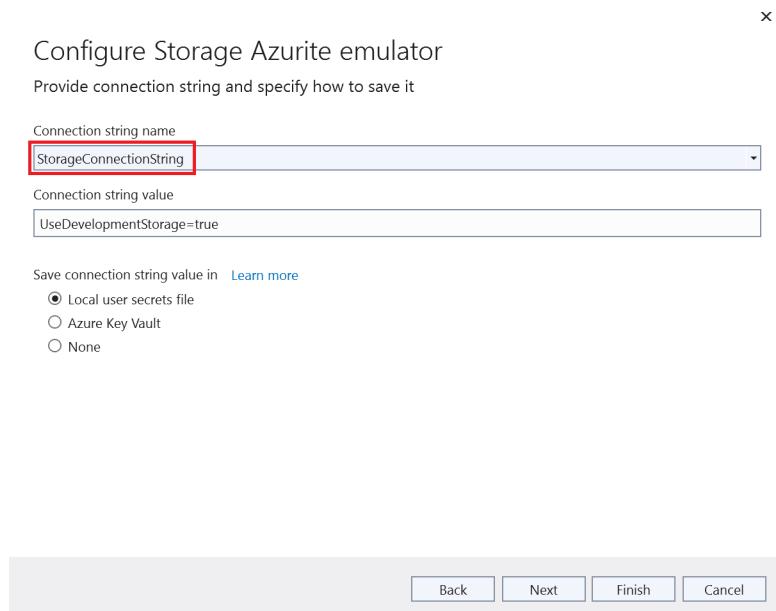
```

#### Running Azurite from an ASP.NET project

In Visual Studio 2022, create an **ASP.NET Core Web App** project. Then, open the **Connected Services** dialog box, select **Add a service dependency**, and then select **Storage Azurite emulator**.



In the **Configure Storage Azurite emulator** dialog box, set the **Connection string name** field to `StorageConnectionString`, and then select **Finish**.



When the configuration completes, select **Close**. The Azurite emulator starts automatically.

```

Output
Show output from: Service Dependencies
WebApplication2: Stopping Azure Storage Azurite emulator...
WebApplication3: Stopping Azure Storage Azurite emulator...
WebApplication3: Azure Blob service is starting at http://127.0.0.1:10000
WebApplication3: Azure Blob service is successfully listening at http://127.0.0.1:10000
WebApplication3: Azure Queue service is starting at http://127.0.0.1:10001
WebApplication3: Azure Queue service is successfully listening at http://127.0.0.1:10001
WebApplication3: Azure Table service is starting at http://127.0.0.1:10002
WebApplication3: Azure Table service is successfully listening at http://127.0.0.1:10002

```

#### Command-line options

This section details the command line switches available when launching Azurite.

##### Help

**Optional** - Get command-line help by using the `-h` or `--help` switch.

```
azurite -h
azurite --help
```

#### Blob listening host

**Optional** - By default, Azurite will listen to 127.0.0.1 as the local server. Use the `--blobHost` switch to set the address to your requirements.

Accept requests on the local machine only:

```
azurite --blobHost 127.0.0.1
```

Allow remote requests:

```
azurite --blobHost 0.0.0.0
```

**Caution**

Allowing remote requests may make your system vulnerable to external attacks.

#### Blob listening port configuration

**Optional** - By default, Azurite will listen for the Blob service on port 10000. Use the `--blobPort` switch to specify the listening port that you require.

**NOTE**

After using a customized port, you need to update the connection string or corresponding configuration in your Azure Storage tools or SDKs.

Customize the Blob service listening port:

```
azurite --blobPort 8888
```

Let the system auto select an available port:

```
azurite --blobPort 0
```

The port in use is displayed during Azurite startup.

#### Queue listening host

**Optional** - By default, Azurite will listen to 127.0.0.1 as the local server. Use the `--queueHost` switch to set the address to your requirements.

Accept requests on the local machine only:

```
azurite --queueHost 127.0.0.1
```

Allow remote requests:

```
azurite --queueHost 0.0.0.0
```

**Caution**

Allowing remote requests may make your system vulnerable to external attacks.

#### Queue listening port configuration

**Optional** - By default, Azurite will listen for the Queue service on port 10001. Use the `--queuePort` switch to specify the listening port that you require.

**NOTE**

After using a customized port, you need to update the connection string or corresponding configuration in your Azure Storage tools or SDKs.

Customize the Queue service listening port:

```
azurite --queuePort 8888
```

Let the system auto select an available port:

```
azurite --queuePort 0
```

The port in use is displayed during Azurite startup.

#### Table listening host

**Optional** - By default, Azurite will listen to 127.0.0.1 as the local server. Use the `--tableHost` switch to set the address to your requirements.

Accept requests on the local machine only:

```
azurite --tableHost 127.0.0.1
```

Allow remote requests:

```
azurite --tableHost 0.0.0.0
```

#### Caution

Allowing remote requests may make your system vulnerable to external attacks.

#### Table listening port configuration

**Optional** - By default, Azurite will listen for the Table service on port 10002. Use the `--tablePort` switch to specify the listening port that you require.

#### NOTE

After using a customized port, you need to update the connection string or corresponding configuration in your Azure Storage tools or SDKs.

Customize the Table service listening port:

```
azurite --tablePort 11111
```

Let the system auto select an available port:

```
azurite --tablePort 0
```

The port in use is displayed during Azurite startup.

#### Workspace path

**Optional** - Azurite stores data to the local disk during execution. Use the `-l` or `--location` switch to specify a path as the workspace location. By default, the current process working directory will be used. Note the lowercase 'l'.

```
azurite -l c:\azurite
azurite --location c:\azurite
```

#### Access log

**Optional** - By default, the access log is displayed in the console window. Disable the display of the access log by using the `-s` or `--silent` switch.

```
azurite -s
azurite --silent
```

#### Debug log

**Optional** - The debug log includes detailed information on every request and exception stack trace. Enable the debug log by providing a valid local file path to the `-d` or `--debug` switch.

```
azurite -d path/debug.log
azurite --debug path/debug.log
```

#### Loose mode

**Optional** - By default, Azurite applies strict mode to block unsupported request headers and parameters. Disable strict mode by using the `-L` or `--loose` switch. Note the capital 'L'.

```
azurite -L
azurite --loose
```

#### Version

**Optional** - Display the installed Azurite version number by using the `-v` or `--version` switch.

```
azurite -v
azurite --version
```

#### Certificate configuration (HTTPS)

**Optional** - By default, Azurite uses the HTTP protocol. Enable HTTPS mode by providing a path to a Privacy Enhanced Mail (.pem) or Personal Information Exchange (.pfx) certificate file to the `--cert` switch.

When `--cert` is provided for a PEM file, you must provide a corresponding `--key` switch.

```
azurite --cert path/server.pem --key path/key.pem
```

When `--cert` is provided for a PFX file, you must provide a corresponding `--pwd` switch.

```
azurite --cert path/server.pfx --pwd pfxpassword
```

For detailed information on creating PEM and PFX files, see [HTTPS Setup](#).

#### OAuth configuration

**Optional** - Enable OAuth authentication for Azurite by using the `--oauth` switch.

```
azurite --oauth basic --cert path/server.pem --key path/key.pem
```

**NOTE**

OAuth requires an HTTPS endpoint. Make sure HTTPS is enabled by providing `--cert` switch along with the `--oauth` switch.

Azurite supports basic authentication by specifying the `basic` parameter to the `--oauth` switch. Azurite will do basic authentication, like validating the incoming bearer token, checking the issuer, audience, and expiry. Azurite won't check the token signature or permissions.

**Skip API Version Check**

**Optional** - When starting up, Azurite checks that the requested API version is valid. The following command skips the API version check:

```
azurite --skipApiVersionCheck
```

**Disable Production Style Url**

**Optional**. When using the fully-qualified domain name instead of the IP in request Uri host, by default Azurite will parse the storage account name from request Uri host. You can force the parsing of the storage account name from request Uri path by using `--disableProductStyleUrl`:

```
azurite --disableProductStyleUrl
```

## Authorization for tools and SDKs

Connect to Azurite from Azure Storage SDKs or tools, like [Azure Storage Explorer](#), by using any authentication strategy. Authentication is required. Azurite supports authorization with OAuth, Shared Key, and shared access signatures (SAS). Azurite also supports anonymous access to public containers.

If you're using the Azure SDKs, start Azurite with the `--oauth basic` and `--cert --key/-pwd` options.

**Well-known storage account and key**

Azurite accepts the same well-known account and key used by the legacy Azure Storage Emulator.

- Account name: `devstoreaccount1`
- Account key: `Eby8vdM02xNocqFlqUwJPLlmEt1CDXJ1OUzFT50uSRZ6IFsuFq2UVeCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==`

**Custom storage accounts and keys**

Azurite supports custom storage account names and keys by setting the `AZURITE_ACCOUNTS` environment variable in the following format: `account1:key1[:key2];account2:key1[:key2];...`.

For example, use a custom storage account that has one key:

```
set AZURITE_ACCOUNTS="account1:key1"
```

```
export AZURITE_ACCOUNTS="account1:key1"
```

**NOTE**

The account keys must be a base64 encoded string.

Or use multiple storage accounts with two keys each:

```
set AZURITE_ACCOUNTS="account1:key1:key2;account2:key1:key2"
```

```
export AZURITE_ACCOUNTS="account1:key1:key2;account2:key1:key2"
```

Azurite refreshes custom account names and keys from the environment variable every minute by default. With this feature, you can dynamically rotate the account key, or add new storage accounts without restarting Azurite.

**NOTE**

The default `devstoreaccount1` storage account is disabled when you set custom storage accounts.

The account keys must be a base64 encoded string.

**Connection strings**

The easiest way to connect to Azurite from your application is to configure a connection string in your application's configuration file that references the shortcut `UseDevelopmentStorage=true`. Here's an example of a connection string in an `app.config` file:

```
<appSettings>
 <add key="StorageConnectionString" value="UseDevelopmentStorage=true" />
</appSettings>
```

**HTTP connection strings**

You can pass the following connection strings to the [Azure SDKs](#) or tools, like Azure CLI 2.0 or Storage Explorer.

The full connection string is:

```
DefaultEndpointsProtocol=http;AccountName=devstoreaccount1;AccountKey=Eby8vdM02xNocqFlqUwJPLlmEt1CDXJ1OUzFT50uSRZ6IFsuFq2UVeCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==;
```

To connect to the blob service only, the connection string is:

```
DefaultEndpointsProtocol=http;AccountName=devstoreaccount1;AccountKey=Eby8vdM02xN0cqFlqUwJPlLmEt1CDXJ10UzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==
```

To connect to the queue service only, the connection string is:

```
DefaultEndpointsProtocol=http;AccountName=devstoreaccount1;AccountKey=Eby8vdM02xN0cqFlqUwJPlLmEt1CDXJ10UzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==
```

To connect to the table service only, the connection string is:

```
DefaultEndpointsProtocol=http;AccountName=devstoreaccount1;AccountKey=Eby8vdM02xN0cqFlqUwJPlLmEt1CDXJ10UzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==
```

#### HTTPS connection strings

The full HTTPS connection string is:

```
DefaultEndpointsProtocol=https;AccountName=devstoreaccount1;AccountKey=Eby8vdM02xN0cqFlqUwJPlLmEt1CDXJ10UzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==
```

To use the blob service only, the HTTPS connection string is:

```
DefaultEndpointsProtocol=https;AccountName=devstoreaccount1;AccountKey=Eby8vdM02xN0cqFlqUwJPlLmEt1CDXJ10UzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==
```

To use the queue service only, the HTTPS connection string is:

```
DefaultEndpointsProtocol=https;AccountName=devstoreaccount1;AccountKey=Eby8vdM02xN0cqFlqUwJPlLmEt1CDXJ10UzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==
```

To use the table service only, the HTTPS connection string is:

```
DefaultEndpointsProtocol=https;AccountName=devstoreaccount1;AccountKey=Eby8vdM02xN0cqFlqUwJPlLmEt1CDXJ10UzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==
```

If you used `dotnet dev-certs` to generate your self-signed certificate, use the following connection string.

```
DefaultEndpointsProtocol=https;AccountName=devstoreaccount1;AccountKey=Eby8vdM02xN0cqFlqUwJPlLmEt1CDXJ10UzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==
```

Update the connection string when using [custom storage accounts and keys](#).

For more information, see [Configure Azure Storage connection strings](#).

#### Azure SDKs

To use Azurite with the [Azure SDKs](#), use OAuth and HTTPS options:

```
azurite --oauth basic --cert certname.pem --key certname-key.pem
```

#### Azure Blob Storage

You can then instantiate a `BlobContainerClient`, `BlobServiceClient`, or `BlobClient`.

```
// With container URL and DefaultAzureCredential
var client = new BlobContainerClient(
 new Uri("https://127.0.0.1:10000/devstoreaccount1/container-name"), new DefaultAzureCredential()
);

// With connection string
var client = new BlobContainerClient(
 "DefaultEndpointsProtocol=https;AccountName=devstoreaccount1;AccountKey=Eby8vdM02xN0cqFlqUwJPlLmEt1CDXJ10UzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==;BlobEndpoint=https://127.0.0.1:10000/devstoreaccount1;",
 "container-name"
);

// With account name and key
var client = new BlobContainerClient(
 new Uri("https://127.0.0.1:10000/devstoreaccount1/container-name"),
 new StorageSharedKeyCredential("devstoreaccount1",
 "Eby8vdM02xN0cqFlqUwJPlLmEt1CDXJ10UzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==")
);
```

#### Azure Queue Storage

You can also instantiate a `QueueClient` or `QueueServiceClient`.

```
// With queue URL and DefaultAzureCredential
var client = new QueueClient(
 new Uri("https://127.0.0.1:10001/devstoreaccount1/queue-name"), new DefaultAzureCredential()
);

// With connection string
var client = new QueueClient(
 "DefaultEndpointsProtocol=https;AccountName=devstoreaccount1;AccountKey=Eby8vdM02xN0cqFlqUwJPlLmEt1CDXJ10UzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==;QueueEndpoint=https://127.0.0.1:10001/devstoreaccount1;",
 "queue-name"
);

// With account name and key
var client = new QueueClient(
 new Uri("https://127.0.0.1:10001/devstoreaccount1/queue-name"),
 new StorageSharedKeyCredential("devstoreaccount1",
 "Eby8vdM02xN0cqFlqUwJPlLmEt1CDXJ10UzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==")
);
```

#### Azure Table Storage

You can also instantiate a `TableClient` or `TableServiceClient`.

```

// With table URL and DefaultAzureCredential
var client = new Client(
 new Uri("https://127.0.0.1:10002/devstoreaccount1/table-name"), new DefaultAzureCredential()
);

// With connection string
var client = new TableClient(
 "DefaultEndpointsProtocol=https;AccountName=devstoreaccount1;AccountKey=Eby8vdM02xNOcqFlqUwJPLmEt1CDXJ1OUzFT50uSRZ6IfsuFq2UVrCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==;TableEndpoint=https://127.0.0.1:10002/devstoreaccount1",
 "table-name"
);

// With account name and key
var client = new TableClient(
 new Uri("https://127.0.0.1:10002/devstoreaccount1/table-name"),
 new StorageSharedKeyCredential("devstoreaccount1",
 "Eby8vdM02XN0cqFlqUwJPLmEt1CDXJ1OUzFT50uSRZ6IfsuFq2UVrCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==")
);

```

### Microsoft Azure Storage Explorer

You can use Storage Explorer to view the data stored in Azurite.

#### Connect to Azurite using HTTP

In Storage Explorer, connect to Azurite by following these steps:

1. Select the **Manage Accounts** icon
2. Select **Add an account**
3. Select **Attach to a local emulator**
4. Select **Next**
5. Edit the **Display name** field to a name of your choice
6. Select **Next** again
7. Select **Connect**

#### Connect to Azurite using HTTPS

By default Storage Explorer won't open an HTTPS endpoint that uses a self-signed certificate. If you're running Azurite with HTTPS, you're likely using a self-signed certificate. In Storage Explorer, import SSL certificates via the **Edit -> SSL Certificates -> Import Certificates** dialog.

#### Import Certificate to Storage Explorer

1. Find the certificate on your local machine.
2. In Storage Explorer, go to **Edit -> SSL Certificates -> Import Certificates** and import your certificate.

If you don't import a certificate, you'll get an error:

```
unable to verify the first certificate | or | self signed certificate in chain
```

#### Add Azurite via HTTPS connection string

Follow these steps to add Azurite HTTPS to Storage Explorer:

1. Select **Toggle Explorer**
2. Select **Local & Attached**
3. Right-click on **Storage Accounts** and select **Connect to Azure Storage**.
4. Select **Use a connection string**
5. Select **Next**.
6. Enter a value in the **Display name** field.
7. Enter the **HTTPS connection string** from the previous section of this document
8. Select **Next**
9. Select **Connect**

## Workspace structure

The following files and folders may be created in the workspace location when initializing Azurite.

- `__blobstorage__` - Directory containing Azurite blob service persisted binary data
- `__queuestorage__` - Directory containing Azurite queue service persisted binary data
- `__tablestorage__` - Directory containing Azurite table service persisted binary data
- `__azure_db_blob__.json` - Azurite blob service metadata file
- `__azure_db_blob_extent__.json` - Azurite blob service extent metadata file
- `__azure_db_queue__.json` - Azurite queue service metadata file
- `__azure_db_queue_extent__.json` - Azurite queue service extent metadata file
- `__azure_db_table__.json` - Azurite table service metadata file
- `__azurite_db_table_extent__.json` - Azurite table service extent metadata file

To clean up Azurite, delete above files and folders and restart the emulator.

## Differences between Azurite and Azure Storage

There are functional differences between a local instance of Azurite and an Azure Storage account in the cloud.

#### Endpoint and connection URL

The service endpoints for Azurite are different from the endpoints of an Azure Storage account. The local computer doesn't do domain name resolution, requiring Azurite endpoints to be local addresses.

When you address a resource in an Azure Storage account, the account name is part of the URI host name. The resource being addressed is part of the URI path:

```
<http|https://<account-name>.<service-name>.core.windows.net/<resource-path>
```

The following URI is a valid address for a blob in an Azure Storage account:

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

#### IP-style URL

Since the local computer doesn't resolve domain names, the account name is part of the URI path instead of the host name. Use the following URI format for a resource in Azurite:

```
http://<local-machine-address>:<port>/<account-name>/<resource-path>
```

The following address might be used for accessing a blob in Azurite:

```
http://127.0.0.1:10000/myaccount/mycontainer/myblob.txt
```

#### Production-style URL

Optionally, you could modify your hosts file to access an account with *production-style URL*.

First, add one or more lines to your hosts file. For example:

```
127.0.0.1 account1.blob.localhost
127.0.0.1 account1.queue.localhost
127.0.0.1 account1.table.localhost
```

Next, set environment variables to enable customized storage accounts and keys:

```
set AZURITE_ACCOUNTS="account1:key1:key2"
```

You could add more accounts. See the [Custom storage accounts and keys](#) section of this article.

Start Azurite and use a customized connection string to access your account. The example connection string below assumes that the default ports are used.

```
DefaultEndpointsProtocol=http;AccountName=account1;AccountKey=key1;BlobEndpoint=http://account1.blob.localhost:10000;QueueEndpoint=http://account1.queue.localhost:10001;TableEndpoint=http://account1.table.localhost:10002;
```

Do not access default account in this way with Azure Storage Explorer. There is a bug that Storage Explorer is always adding account name in URL path, causing failures.

By default, when using Azurite with a production-style URL, the account name should be the host name in fully-qualified domain name such as "http://devstoreaccount1.blob.localhost:10000/container". To use production-style URL with account name in the URL path such as "http://foo.bar.com:10000/devstoreaccount1/container", make sure to use the `--disableProductStyleUrl` parameter when you start Azurite.

If use `host.docker.internal` as request Uri host (For example:

```
http://host.docker.internal:10000/devstoreaccount1/container
```

Azurite will always get the account name from the request Uri path. This is true regardless of whether you use the `--disableProductStyleUrl` parameter when you start Azurite.

#### Scaling and performance

Azurite doesn't support large numbers of connected clients. There's no performance guarantee. Azurite is intended for development and testing purposes.

#### Error handling

Azurite is aligned with Azure Storage error handling logic, but there are differences. For example, error messages may be different, while error status codes align.

#### RA-GRS

Azurite supports read-access geo-redundant replication (RA-GRS). For storage resources, access the secondary location by appending `-secondary` to the account name. For example, the following address might be used for accessing a blob using the read-only secondary in Azurite:

```
http://127.0.0.1:10000/devstoreaccount1-secondary/mycontainer/myblob.txt
```

#### Table support

Support for tables in Azurite is currently in preview. For more information, see the [Azurite V3 Table](#) project.

Support for durable functions requires tables.

#### IMPORTANT

Azurite support for Table Storage is currently in **PREVIEW**. See the [Supplemental Terms of Use for Microsoft Azure Previews](#) for legal terms that apply to Azure features that are in beta, preview, or otherwise not yet released into general availability.

## Azurite is open-source

Contributions and suggestions for Azurite are welcome. Go to the Azurite [GitHub project](#) page or [GitHub issues](#) for milestones and work items we're tracking for upcoming features and bug fixes. Detailed work items are also tracked in GitHub.

## Next steps

- [Use the Azure Storage Emulator for development and testing](#) documents the legacy Azure Storage Emulator, which is being superseded by Azurite.
- [Configure Azure Storage connection strings](#) explains how to assemble a valid Azure Storage connection string.

# Run automated tests by using Azurite

8/22/2022 • 2 minutes to read • [Edit Online](#)

Learn how to write automated tests against private endpoints for Azure Blob Storage by using the Azurite storage emulator.

## Run tests on your local machine

1. Install the latest version of [Python](#)

2. Install [Azure Storage Explorer](#)

3. Install and run [Azurite](#):

**Option 1:** Use npm to install, then run Azurite locally

```
Install Azurite
npm install -g azurite

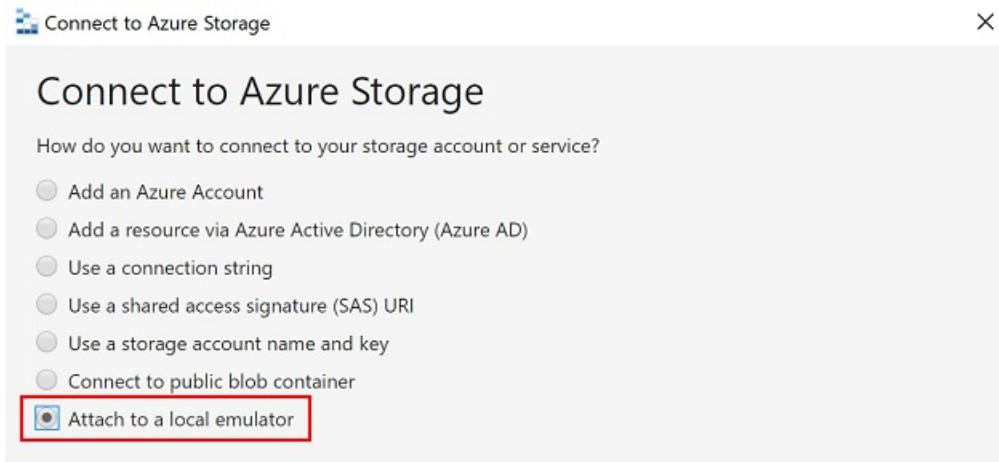
Create an Azurite directory
mkdir c:\azurite

Launch Azurite locally
azurite --silent --location c:\azurite --debug c:\azurite\debug.log
```

**Option 2:** Use Docker to run Azurite

```
docker run -p 10000:10000 mcr.microsoft.com/azure-storage/azurite azurite-blob --blobHost 0.0.0.0
```

4. In Azure Storage Explorer, select **Attach to a local emulator**



5. Provide a **Display name** and **Blobs port** number to connect Azurite and use Azure Storage Explorer to manage local blob storage.

## Attach to Local Emulator

Display name:

Protocol:



Blobs port:

Files port:

Queues port:

Tables port:

### 6. Create a virtual Python environment

```
python -m venv .venv
```

### 7. Create a container and initialize environment variables. Use a [PyTest conftest.py](#) file to generate tests.

Here is an example of a conftest.py file:

```
from azure.storage.blob import BlobServiceClient
import os

def pytest_generate_tests(metafunc):
 os.environ['AZURE_STORAGE_CONNECTION_STRING'] =
 'DefaultEndpointsProtocol=http;AccountName=devstoreaccount1;AccountKey=Eby8vdM02xN0cqFlqUwJPLlmEt1CDX
 J1OUzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPTOtr/KBHBeksoGMGw==;BlobEndpoint=http://127.0.0.1:10000/devsto
 reaccount1;'
 os.environ['STORAGE_CONTAINER'] = 'test-container'

 # Create a container for Azurite for the first run
 blob_service_client =
 BlobServiceClient.from_connection_string(os.environ.get("AZURE_STORAGE_CONNECTION_STRING"))
 try:
 blob_service_client.create_container(os.environ.get("STORAGE_CONTAINER"))
 except Exception as e:
 print(e)
```

## NOTE

The value shown for `AZURE_STORAGE_CONNECTION_STRING` is the default value for Azurite, it's not a private key.

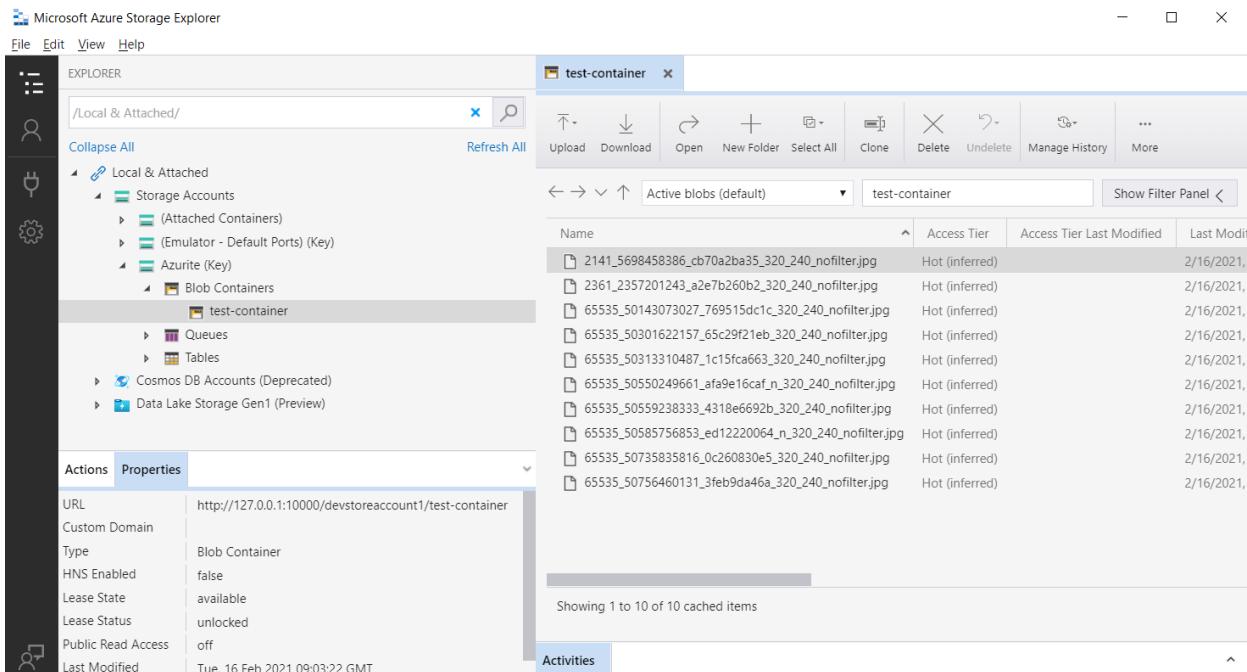
## 8. Install dependencies listed in a `requirements.txt` file

```
pip install -r requirements.txt
```

## 9. Run tests:

```
python -m pytest ./tests
```

After running tests, you can see the files in Azurite blob storage by using Azure Storage Explorer.



## Run tests on Azure Pipelines

After running tests locally, make sure the tests pass on [Azure Pipelines](#). Use a Docker Azurite image as a hosted agent on Azure, or use npm to install Azurite. The following Azure Pipelines example uses npm to install Azurite.

```

trigger:
- master

steps:
- task: UsePythonVersion@0
 displayName: 'Use Python 3.7'
 inputs:
 versionSpec: 3.7

- bash: |
 pip install -r requirements_tests.txt
 displayName: 'Setup requirements for tests'

- bash: |
 sudo npm install -g azurite
 sudo mkdir azurite
 sudo azurite --silent --location azurite --debug azurite\debug.log &
 displayName: 'Install and Run Azurite'

- bash: |
 python -m pytest --junit-xml=unit_tests_report.xml --cov=tests --cov-report=html --cov-report=xml
./tests
 displayName: 'Run Tests'

- task: PublishCodeCoverageResults@1
 inputs:
 codeCoverageTool: Cobertura
 summaryFileLocation: '$(System.DefaultWorkingDirectory)/**/coverage.xml'
 reportDirectory: '$(System.DefaultWorkingDirectory)/**/htmlcov'

- task: PublishTestResults@2
 inputs:
 testResultsFormat: 'JUnit'
 testResultsFiles: '**/*_tests_report.xml'
 failTaskOnFailedTests: true

```

After running the Azure Pipelines tests, you should see output similar to this:

The screenshot shows the Azure Pipelines interface. On the left, the 'Jobs in run #20210218.3' section is visible, listing various tasks such as 'Job', 'Initialize job', 'Checkout Azurite PyTest Sample', 'Use Python 3.7', 'Setup requirements for tests', 'Install and Run Azurite', 'Run Tests', 'PublishCodeCoverageResults', 'PublishTestResults', 'Component Detection (auto-inje...)', 'Post-job: Checkout Azurite PyTe...', 'Finalize Job', and 'Report build status'. Most of these tasks have a green checkmark indicating they were successful. On the right, a detailed log for the 'Run Tests' task is displayed. The log shows the command being run: 'python -m pytest --junit-xml=unit\_tests\_report.xml --cov=tests --cov-report=html --cov-report=xml ./tests'. It also shows the command output, which includes generating a script, starting the command output, and the test session starts. The log concludes with 'Finishing: Run Tests'.

## Next steps

- Use the Azurite emulator for local Azure Storage development
- Sample: Using Azurite to run blob storage tests in Azure DevOps Pipeline

# Use the Azure Storage Emulator for development and testing (deprecated)

8/22/2022 • 16 minutes to read • [Edit Online](#)

The Microsoft Azure Storage Emulator is a tool that emulates the Azure Blob, Queue, and Table services for local development purposes. You can test your application against the storage services locally without creating an Azure subscription or incurring any costs. When you're satisfied with how your application is working in the emulator, switch to using an Azure storage account in the cloud.

## IMPORTANT

The Azure Storage Emulator is now deprecated. Microsoft recommends that you use the [Azurite](#) emulator for local development with Azure Storage. Azurite supersedes the Azure Storage Emulator. Azurite will continue to be updated to support the latest versions of Azure Storage APIs. For more information, see [Use the Azurite emulator for local Azure Storage development](#).

## Get the Storage Emulator

The Storage Emulator is available as part of the [Microsoft Azure SDK](#). You can also install the Storage Emulator by using the [standalone installer](#) (direct download). To install the Storage Emulator, you must have administrative privileges on your computer.

The Storage Emulator currently runs only on Windows. For emulation on Linux, use the [Azurite](#) emulator.

## NOTE

Data created in one version of the Storage Emulator is not guaranteed to be accessible when using a different version. If you need to persist your data for the long term, we recommend that you store that data in an Azure storage account, rather than in the Storage Emulator.

The Storage Emulator depends on specific versions of the OData libraries. Replacing the OData DLLs used by the Storage Emulator with other versions is unsupported, and may cause unexpected behavior. However, any version of OData supported by the storage service may be used to send requests to the emulator.

## How the storage emulator works

The Storage Emulator uses a local Microsoft SQL Server 2012 Express LocalDB instance to emulate Azure storage services. You can choose to configure the Storage Emulator to access a local instance of SQL Server instead of the LocalDB instance. See the [Start and initialize the Storage Emulator](#) section later in this article to learn more.

The Storage Emulator connects to SQL Server or LocalDB using Windows authentication.

Some differences in functionality exist between the Storage Emulator and Azure storage services. For more information about these differences, see the [Differences between the Storage Emulator and Azure Storage](#) section later in this article.

## Start and initialize the Storage Emulator

To start the Azure Storage Emulator:

1. Select the **Start** button or press the **Windows** key.
2. Begin typing `Azure Storage Emulator`.
3. Select the emulator from the list of displayed applications.

When the Storage Emulator starts, a Command Prompt window will appear. You can use this console window to start and stop the Storage Emulator. You can also clear data, get status, and initialize the emulator from the command prompt. For more information, see the [Storage Emulator command-line tool reference](#) section later in this article.

**NOTE**

The Azure Storage Emulator may not start correctly if another storage emulator, such as Azurite, is running on the system.

When the emulator is running, you'll see an icon in the Windows taskbar notification area.

When you close the Storage Emulator Command Prompt window, the Storage Emulator will continue to run. To bring up the Storage Emulator console window again, follow the preceding steps as if starting the Storage Emulator.

The first time you run the Storage Emulator, the local storage environment is initialized for you. The initialization process creates a database in LocalDB and reserves HTTP ports for each local storage service.

The Storage Emulator is installed by default to `C:\Program Files (x86)\Microsoft SDKs\Azure\Storage Emulator`.

**TIP**

You can use the [Microsoft Azure Storage Explorer](#) to work with local Storage Emulator resources. Look for "(Emulator - Default Ports) (Key)" under "Local & Attached" in the Storage Explorer resources tree after you've installed and started the Storage Emulator.

### Initialize the storage Emulator to use a different SQL database

You can use the storage Emulator command-line tool to initialize the Storage Emulator to point to a SQL database instance other than the default LocalDB instance:

1. Open the Storage Emulator console window as described in the [Start and initialize the Storage Emulator](#) section.
2. In the console window, type the following command, where `<SQLServerInstance>` is the name of the SQL Server instance. To use LocalDB, specify `(localdb)\MSSQLLocalDb` as the SQL Server instance.

```
AzureStorageEmulator.exe init /server <SQLServerInstance>
```

You can also use the following command, which directs the emulator to use the default SQL Server instance:

```
AzureStorageEmulator.exe init /server .
```

Or, you can use the following command, which initializes the database to the default LocalDB instance:

```
AzureStorageEmulator.exe init /forceCreate
```

For more information about these commands, see [Storage Emulator command-line tool reference](#).

**TIP**

You can use the Microsoft SQL Server Management Studio (SSMS) to manage your SQL Server instances, including the LocalDB installation. In the SMSS **Connect to Server** dialog, specify `(localdb)\MSSQLLocalDb` in the **Server name:** field to connect to the LocalDB instance.

## Authenticating requests against the Storage Emulator

Once you've installed and started the Storage Emulator, you can test your code against it. Every request you make against the Storage Emulator must be authorized, unless it's an anonymous request. You can authorize requests against the Storage Emulator using Shared Key authentication or with a shared access signature (SAS).

### Authorize with Shared Key credentials

The emulator supports a single fixed account and a well-known authentication key for Shared Key authentication. This account and key are the only Shared Key credentials permitted for use with the emulator. They are:

```
Account name: devstoreaccount1
Account key: Eby8vdM02xN0cqFlqUwJPLl1mEt1CDXJ10UzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==
```

**NOTE**

The authentication key supported by the emulator is intended only for testing the functionality of your client authentication code. It does not serve any security purpose. You cannot use your production storage account and key with the emulator. You should not use the development account with production data.

The emulator supports connection via HTTP only. However, HTTPS is the recommended protocol for accessing resources in a production Azure storage account.

### Connect to the emulator account using the shortcut

The easiest way to connect to the emulator from your application is to configure a connection string in your application's configuration file that references the shortcut `UseDevelopmentStorage=true`. The shortcut is equivalent to the full connection string for the emulator, which specifies the account name, the account key, and the emulator endpoints for each of the Azure Storage services:

```
DefaultEndpointsProtocol=http;AccountName=devstoreaccount1;
AccountKey=Eby8vdM02xN0cqFlqUwJPLl1mEt1CDXJ10UzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==;
BlobEndpoint=http://127.0.0.1:10000/devstoreaccount1;
QueueEndpoint=http://127.0.0.1:10001/devstoreaccount1;
TableEndpoint=http://127.0.0.1:10002/devstoreaccount1;
```

The following .NET code snippet shows how you can use the shortcut from a method that takes a connection string. For example, the `BlobContainerClient(String, String)` constructor takes a connection string.

```
BlobContainerClient blobContainerClient = new BlobContainerClient("UseDevelopmentStorage=true", "sample-
container");
blobContainerClient.CreateIfNotExists();
```

Make sure that the emulator is running before calling the code in the snippet.

For more information on connection strings, see [Configure Azure Storage connection strings](#).

### Authorize with a shared access signature

#### NOTE

This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

Some Azure storage client libraries, such as the Xamarin library, only support authentication with a shared access signature (SAS) token. You can create the SAS token using [Storage Explorer](#) or another application that supports Shared Key authentication.

You can also generate a SAS token by using Azure PowerShell. The following example generates a SAS token with full permissions to a blob container:

1. Install Azure PowerShell if you haven't already (using the latest version of the Azure PowerShell cmdlets is recommended). For installation instructions, see [Install and configure Azure PowerShell](#).
2. Open Azure PowerShell and run the following commands, replacing `CONTAINER_NAME` with a name of your choosing:

```
$context = New-AzStorageContext -Local

New-AzStorageContainer CONTAINER_NAME -Permission Off -Context $context

$now = Get-Date

New-AzStorageContainerSASToken -Name CONTAINER_NAME -Permission rwdl -ExpiryTime $now.AddDays(1.0) -Context
$context -FullUri
```

The resulting shared access signature URI for the new container should be similar to:

```
http://127.0.0.1:10000/devstoreaccount1/sascontainer?sv=2012-02-12&se=2015-07-
08T00%3A12%3A08Z&sr=c&sp=w&sig=t%2BbzU9%2B7ry4okULN9S0wst%2F8MCUhTjrHyV9rDNLSe8g%3Dsss
```

The shared access signature created with this example is valid for one day. The signature grants full access (read, write, delete, list) to blobs within the container.

For more information on shared access signatures, see [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#).

## Addressing resources in the Storage Emulator

The service endpoints for the Storage Emulator are different from the endpoints for an Azure storage account. The local computer doesn't do domain name resolution, requiring the Storage Emulator endpoints to be local addresses.

When you address a resource in an Azure storage account, you use the following scheme. The account name is part of the URI host name, and the resource being addressed is part of the URI path:

```
<http|https>://<account-name>.<service-name>.core.windows.net/<resource-path>
```

For example, the following URI is a valid address for a blob in an Azure storage account:

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

Because the local computer doesn't do domain name resolution, the account name is part of the URI path instead of the host name. Use the following URI format for a resource in the Storage Emulator:

```
http://<local-machine-address>:<port>/<account-name>/<resource-path>
```

For example, the following address might be used for accessing a blob in the Storage Emulator:

```
http://127.0.0.1:10000/myaccount/mycontainer/myblob.txt
```

The service endpoints for the Storage Emulator are:

- Blob service: `http://127.0.0.1:10000/<account-name>/<resource-path>`
- Queue service: `http://127.0.0.1:10001/<account-name>/<resource-path>`
- Table service: `http://127.0.0.1:10002/<account-name>/<resource-path>`

### Addressing the account secondary with RA-GRS

Beginning with version 3.1, the Storage Emulator supports read-access geo-redundant replication (RA-GRS). You can access the secondary location by appending -secondary to the account name. For example, the following address might be used for accessing a blob using the read-only secondary in the Storage Emulator:

```
http://127.0.0.1:10000/myaccount-secondary/mycontainer/myblob.txt
```

#### NOTE

For programmatic access to the secondary with the Storage Emulator, use the Storage Client Library for .NET version 3.2 or later. See the [Microsoft Azure Storage Client Library for .NET](#) for details.

## Storage Emulator command-line tool reference

Starting in version 3.0, a console window is displayed when you start the Storage Emulator. Use the command line in the console window to start and stop the emulator. You can also query for status and do other operations from the command line.

#### NOTE

If you have the Microsoft Azure Compute Emulator installed, a system tray icon appears when you launch the Storage Emulator. Right-click on the icon to reveal a menu that provides a graphical way to start and stop the Storage Emulator.

### Command-line syntax

```
AzureStorageEmulator.exe [start] [stop] [status] [clear] [init] [help]
```

### Options

To view the list of options, type `/help` at the command prompt.

OPTION	DESCRIPTION	COMMAND	ARGUMENTS
Start	Starts up the Storage Emulator.	<code>AzureStorageEmulator.exe start [-inprocess]</code>	<code>-Reprocess</code> : Start the emulator in the current process instead of creating a new process.
Stop	Stops the Storage Emulator.	<code>AzureStorageEmulator.exe stop</code>	
Status	Prints the status of the Storage Emulator.	<code>AzureStorageEmulator.exe status</code>	

OPTION	DESCRIPTION	COMMAND	ARGUMENTS
Clear	Clears the data in all services specified on the command line.	AzureStorageEmulator.exe clear [ <i>blob</i> ] [ <i>table</i> ] [ <i>queue</i> ] [ <i>all</i> ]	<i>blob</i> : Clears blob data. <i>queue</i> : Clears queue data. <i>table</i> : Clears table data. <i>all</i> : Clears all data in all services.
Init	Does one-time initialization to set up the emulator.	AzureStorageEmulator.exe init [-server serverName] [- sqlinstance instanceName] [- forcecreate -skipcreate] [-reserveports - unreserveports] [- inprocess]	- <i>server</i> <i>serverName instanceName</i> : Specifies the server hosting the SQL instance. - <i>sqlinstance instanceName</i> : Specifies the name of the SQL instance to be used in the default server instance. - <i>forcecreate</i> : Forces creation of the SQL database, even if it already exists. - <i>skipcreate</i> : Skips creation of the SQL database. This takes precedence over -forcecreate. - <i>reserveports</i> : Attempts to reserve the HTTP ports associated with the services. - <i>unreserveports</i> : Attempts to remove reservations for the HTTP ports associated with the services. This takes precedence over -reserveports. - <i>inprocess</i> : Performs initialization in the current process instead of spawning a new process. The current process must be launched with elevated permissions if changing port reservations.

## Differences between the Storage Emulator and Azure Storage

Because the Storage Emulator is a local emulated environment, there are differences between using the emulator and an Azure storage account in the cloud:

- The Storage Emulator supports only a single fixed account and a well-known authentication key.
- The Storage Emulator isn't a scalable storage service and doesn't support a large number of concurrent clients.
- As described in [Addressing resources in the Storage Emulator](#), resources are addressed differently in the Storage Emulator versus an Azure storage account. The difference is because domain name resolution is available in the cloud but not on the local computer.
- Beginning with version 3.1, the Storage Emulator account supports read-access geo-redundant replication (RA-GRS). In the emulator, all accounts have RA-GRS enabled and there's never any lag between the primary and secondary replicas. The Get Blob Service Stats, Get Queue Service Stats, and Get Table Service Stats operations are supported on the account secondary and will always return the value of the *LastSyncTime* response element as the current time according to the underlying SQL database.
- The File service and SMB protocol service endpoints aren't currently supported in the Storage Emulator.
- If you use a version of the storage services that is not supported by the emulator, the emulator returns a

VersionNotSupportedByEmulator error (HTTP status code 400 - Bad Request).

## Differences for Blob storage

The following differences apply to Blob storage in the emulator:

- The Storage Emulator only supports blob sizes up to 2 GB.
- The maximum length of a blob name in the Storage Emulator is 256 characters, while the maximum length of a blob name in Azure Storage is 1024 characters.
- Incremental copy allows snapshots from overwritten blobs to be copied, which returns a failure on the service.
- Get Page Ranges Diff doesn't work between snapshots copied using Incremental Copy Blob.
- A Put Blob operation may succeed against a blob that exists in the Storage Emulator with an active lease even if the lease ID hasn't been specified in the request.
- Append blob operations are not supported by the emulator. Attempting an operation on an append blob returns a FeatureNotSupportedByEmulator error (HTTP status code 400 - Bad Request).

## Differences for Table storage

The following differences apply to Table storage in the emulator:

- Date properties in the Table service in the Storage Emulator support only the range supported by SQL Server 2005 (they're required to be later than January 1, 1753). All dates before January 1, 1753 are changed to this value. The precision of dates is limited to the precision of SQL Server 2005, meaning that dates are precise to 1/300th of a second.
- The Storage Emulator supports partition key and row key property values of less than 512 bytes each. The total size of the account name, table name, and key property names together can't exceed 900 bytes.
- The total size of a row in a table in the Storage Emulator is limited to less than 1 MB.
- In the Storage Emulator, properties of data type `Edm.Guid` or `Edm.Binary` support only the `Equal (eq)` and `NotEqual (ne)` comparison operators in query filter strings.

## Differences for Queue storage

There are no differences specific to Queue storage in the emulator.

# Storage Emulator release notes

## Version 5.10

- The Storage Emulator won't reject version 2019-07-07 of the storage services on Blob, Queue, and Table service endpoints.

## Version 5.9

- The Storage Emulator won't reject version 2019-02-02 of the storage services on Blob, Queue, and Table service endpoints.

## Version 5.8

- The Storage Emulator won't reject version 2018-11-09 of the storage services on Blob, Queue, and Table service endpoints.

## Version 5.7

- Fixed a bug that would cause a crash if logging was enabled.

## Version 5.6

- The Storage Emulator now supports version 2018-03-28 of the storage services on Blob, Queue, and Table service endpoints.

## Version 5.5

- The Storage Emulator now supports version 2017-11-09 of the storage services on Blob, Queue, and Table service endpoints.
- Support has been added for the blob **Created** property, which returns the blob's creation time.

#### **Version 5.4**

- To improve installation stability, the emulator no longer attempts to reserve ports at install time. If you want port reservations, use the `-reserveports` option of the `init` command to specify them.

#### **Version 5.3**

- The Storage Emulator now supports version 2017-07-29 of the storage services on Blob, Queue, and Table service endpoints.

#### **Version 5.2**

- The Storage Emulator now supports version 2017-04-17 of the storage services on Blob, Queue, and Table service endpoints.
- Fixed a bug where table property values weren't being properly encoded.

#### **Version 5.1**

- Fixed a bug where the Storage Emulator was returning the `DataServiceVersion` header in some responses where the service was not.

#### **Version 5.0**

- The Storage Emulator installer no longer checks for existing MSSQL and .NET Framework installs.
- The Storage Emulator installer no longer creates the database as part of install. Database will still be created if needed as part of startup.
- Database creation no longer requires elevation.
- Port reservations are no longer needed for startup.
- Adds the following options to `init` : `-reserveports` (requires elevation), `-unreserveports` (requires elevation), `-skipcreate`.
- The Storage Emulator UI option on the system tray icon now launches the command-line interface. The old GUI is no longer available.
- Some DLLs have been removed or renamed.

#### **Version 4.6**

- The Storage Emulator now supports version 2016-05-31 of the storage services on Blob, Queue, and Table service endpoints.

#### **Version 4.5**

- Fixed a bug that caused installation and initialization to fail when the backing database is renamed.

#### **Version 4.4**

- The Storage Emulator now supports version 2015-12-11 of the storage services on Blob, Queue, and Table service endpoints.
- The Storage Emulator's garbage collection of blob data is now more efficient when dealing with large numbers of blobs.
- Fixed a bug that caused container ACL XML to be validated slightly differently from how the storage service does it.
- Fixed a bug that sometimes caused max and min DateTime values to be reported in the incorrect time zone.

#### **Version 4.3**

- The Storage Emulator now supports version 2015-07-08 of the storage services on Blob, Queue, and Table service endpoints.

## Version 4.2

- The Storage Emulator now supports version 2015-04-05 of the storage services on Blob, Queue, and Table service endpoints.

## Version 4.1

- The Storage Emulator now supports version 2015-02-21 of the storage services on Blob, Queue, and Table service endpoints. It doesn't support the new append blob features.
- The emulator now returns a meaningful error message for unsupported versions of storage services. We recommend using the latest version of the emulator. If you get a VersionNotSupportedByEmulator error (HTTP status code 400 - Bad Request), download the latest version of the emulator.
- Fixed a bug wherein a race condition caused table entity data to be incorrect during concurrent merge operations.

## Version 4.0

- The Storage Emulator executable has been renamed to *AzureStorageEmulator.exe*.

## Version 3.2

- The Storage Emulator now supports version 2014-02-14 of the storage services on Blob, Queue, and Table service endpoints. File service endpoints aren't currently supported in the Storage Emulator. See [Versioning for the Azure Storage Services](#) for details about version 2014-02-14.

## Version 3.1

- Read-access geo-redundant storage (RA-GRS) is now supported in the Storage Emulator. The [Get Blob Service Stats](#), [Get Queue Service Stats](#), and [Get Table Service Stats](#) APIs are supported for the account secondary and will always return the value of the LastSyncTime response element as the current time according to the underlying SQL database. For programmatic access to the secondary with the Storage Emulator, use the Storage Client Library for .NET version 3.2 or later. See the Microsoft Azure Storage Client Library for .NET Reference for details.

## Version 3.0

- The Azure Storage Emulator is no longer shipped in the same package as the compute emulator.
- The Storage Emulator graphical user interface is deprecated. It has been replaced by a scriptable command-line interface. For details on the command-line interface, see [Storage Emulator Command-Line Tool Reference](#). The graphical interface will continue to be present in version 3.0, but it can only be accessed when the Compute Emulator is installed by right-clicking on the system tray icon and selecting Show Storage Emulator UI.
- Version 2013-08-15 of the Azure storage services is now fully supported. (Previously this version was only supported by Storage Emulator version 2.2.1 Preview.)

## Next steps

- Evaluate the cross-platform, community-maintained open-source Storage Emulator [Azurite](#).
- [Azure Storage samples using .NET](#) contains links to several code samples you can use when developing your application.
- You can use the [Microsoft Azure Storage Explorer](#) to work with resources in your cloud Storage account, and in the Storage Emulator.

## See Also

- [Local Azure Storage Development with Azurite, Azure SDKs, and Azure Storage Explorer](#)

# Best practices for monitoring Azure Blob Storage

8/22/2022 • 8 minutes to read • [Edit Online](#)

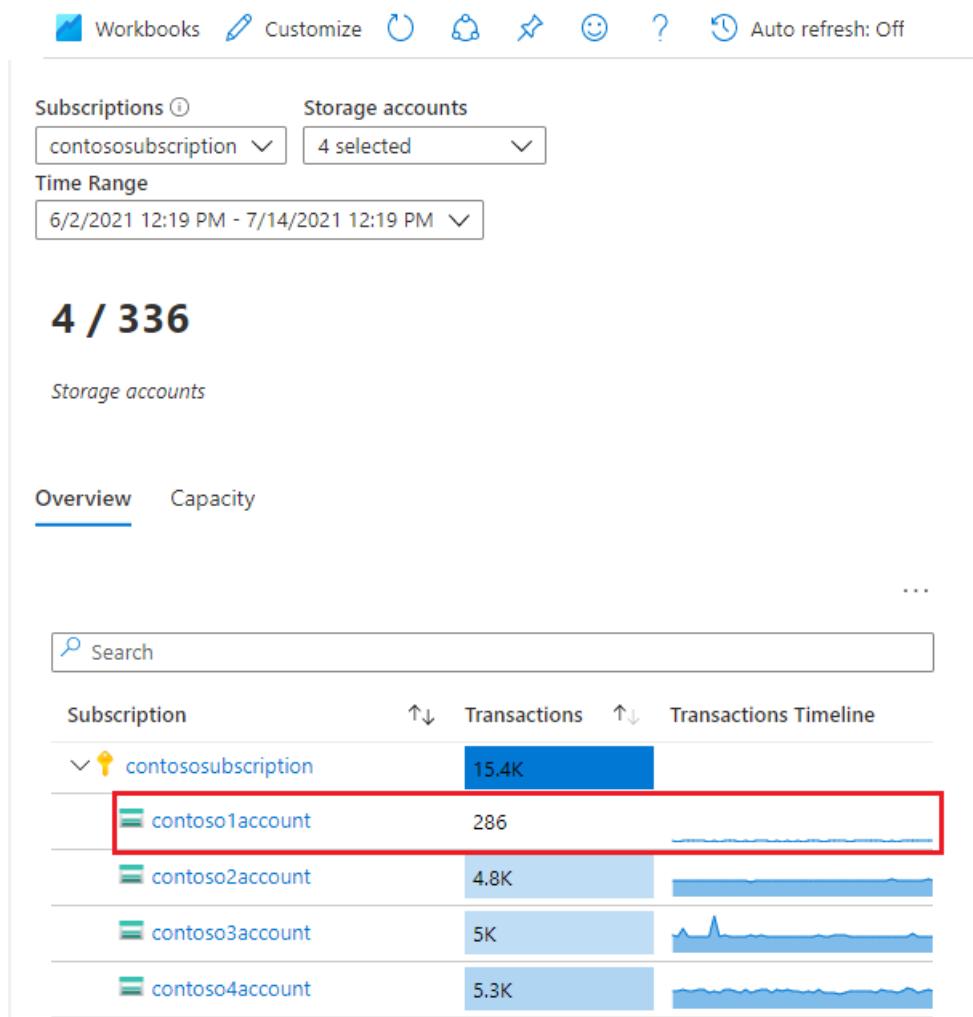
This article features a collection of common storage monitoring scenarios, and provides you with best practice guidelines to accomplish them.

## Identify storage accounts with no or low use

Storage Insights is a dashboard on top of Azure Storage metrics and logs. You can use Storage Insights to examine the transaction volume and used capacity of all your accounts. That information can help you decide which accounts you might want to retire. To configure Storage Insights, see [Monitoring your storage service with Azure Monitor Storage insights](#).

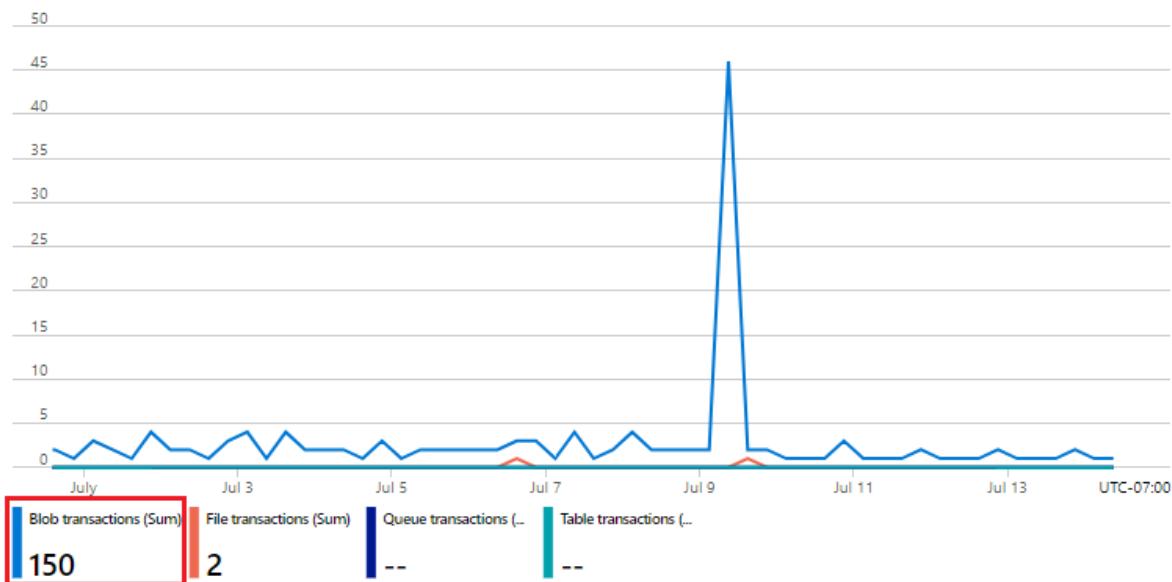
### Analyze transaction volume

From the [Storage Insights view in Azure monitor](#), sort your accounts in ascending order by using the **Transactions** column. The following image shows an account with low transaction volume over the specified period.

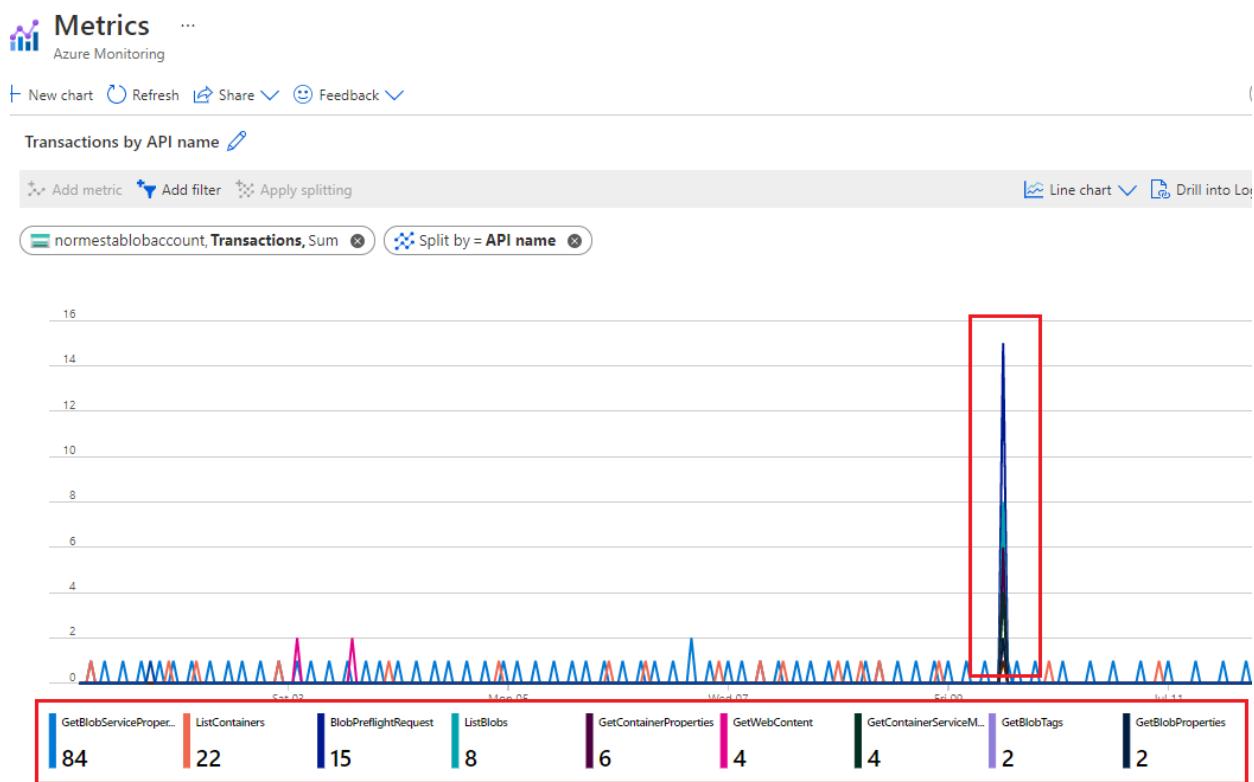


Click the account link to learn more about these transactions. In this example, most requests are made to the Blob Storage service.

## Transactions by storage type



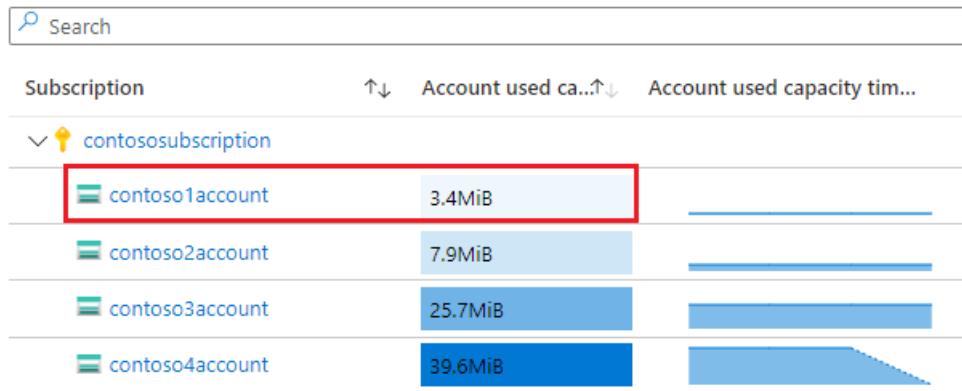
To determine what sorts of requests are being made, drill into the **Transactions by API name** chart.



In this example, all requests are listing operations or requests for account property information. There are no read and write transactions. This might lead you to believe that the account is not being used in a significant way.

## Analyze used capacity

From the **Capacity** tab of the [Storage Insights view in Azure monitor](#), sort your accounts in ascending order by using the **Account used capacity** column. The following image shows an account with lower capacity volume than other accounts.



To examine the blobs associated with this used capacity, you can use Storage Explorer. For large numbers of blobs, consider generating a report by using a [Blob Inventory policy](#).

## Monitor the use of a container

If you partition your customer's data by container, then can monitor how much capacity is used by each customer. You can use Azure Storage blob inventory to take an inventory of blobs with size information. Then, you can aggregate the size and count at the container level. For an example, see [Calculate blob count and total size per container using Azure Storage inventory](#).

You can also evaluate traffic at the container level by querying logs. To learn more about writing Log Analytic queries, see [Log Analytics](#). To learn more about the storage logs schema, see [Azure Blob Storage monitoring data reference](#).

Here's a query to get the number of read transactions and the number of bytes read on each container.

```
StorageBlobLogs
| where OperationName == "GetBlob"
| extend ContainerName = split(parse_url(Uri).Path, "/")[1]
| summarize ReadSize = sum(ResponseBodySize), ReadCount = count() by tostring(ContainerName)
```

The following query uses a similar query to obtain information about write operations.

```
StorageBlobLogs
| where OperationName == "PutBlob" or
 OperationName == "PutBlock" or
 OperationName == "PutBlockList" or
 OperationName == "AppendBlock" or
 OperationName == "SnapshotBlob" or
 OperationName == "CopyBlob" or
 OperationName == "SetBlobTier"
| extend ContainerName = split(parse_url(Uri).Path, "/")[1]
| summarize WriteSize = sum(RequestBodySize), WriteCount = count() by tostring(ContainerName)
```

The above query references the names of multiple operations because more than one type of operation can count as a write operation. To learn more about which operations are considered read and write operations, see either [Azure Blob Storage pricing](#) or [Azure Data Lake Storage pricing](#).

# Audit account activity

In many cases, you'll need to audit the activities of your storage accounts for security and compliance.

Operations on storage accounts fall into two categories: *Control Plane* and *Data Plane*.

A control plane operation is any Azure Resource Manager request to create a storage account or to update a property of an existing storage account. For more information, see [Azure Resource Manager](#).

A data plane operation is an operation on the data in a storage account that results from a request to the storage service endpoint. For example, a data plane operation is executed when you upload a blob to a storage account or download a blob from a storage account. For more information, see [Azure Storage API](#).

The section shows you how to identify the "when", "who", "what" and "how" information of control and data plane operations.

## Auditing control plane operations

Resource Manager operations are captured in the [Azure activity log](#). To view the activity log, open your storage account in the Azure portal, and then select **Activity log**.

Microsoft Azure

Search resources, services, and docs (G+/-)

Home > contoso

contoso | Activity log

Storage account

Search (Ctrl+ /)

Activity Edit columns Refresh Diagnostics settings

Overview

Activity log (selected)

Tags

Diagnose and solve problems

Access Control (IAM)

Data migration

Events

Storage Explorer (preview)

Management Group : None Subscription : contososubscription

Resource group : contosoresourcegroup Resource : contoso

Quick Insights

5 items.

Operation name	Status	Time	Time stamp
List Storage Account	Succeeded	34 minutes ...	Tue Jul 27 2...
List Storage Account	Started	34 minutes ...	Tue Jul 27 2...
List Storage Account	Succeeded	34 minutes ...	Tue Jul 27 2...
List Storage Account	Succeeded	3 hours ago	Tue Jul 27 2...
List Storage Account	Succeeded	3 hours ago	Tue Jul 27 2...

Open any log entry to view JSON that describes the activity. The following JSON shows the "when", "what" and "how" information of a control plane operation:

## List Storage Account Keys

Tue Jul 27 2021 13:38:41 GMT-0700 (Pacific Daylight Time)

+ New alert rule

Summary    **JSON**    Change history (Preview)

```
49 "localizedValue": "Administrative"
50 },
51 "eventTimestamp": "2021-07-27T20:38:41.0356375Z",
52 "id": "/subscriptions/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxx/resourceGroups/
contosoresourcegroup/providers/Microsoft.Storage/storageAccounts/contoso/
events/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxx/ticks/637630151210356375",
53 "level": "Informational",
54 "operationId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
55 "operationName": {
56 "value": "Microsoft.Storage/storageAccounts/listKeys/action",
57 "localizedValue": "List Storage Account Keys"
58 },
59 "resourceGroupName": "contosoresourcegroup",
60 "resourceProviderName": {
61 "value": "Microsoft.Storage",
62 "localizedValue": "Microsoft.Storage"
63 },
64 "resourceType": {
65 "value": "Microsoft.Storage/storageAccounts",
66 "localizedValue": "Microsoft.Storage/storageAccounts"
67 },
68 "resourceId": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx97/resourceGroups/
contosoresourcegroup/providers/Microsoft.Storage/storageAccounts/contoso",
-- " . . . "
```

The availability of the "who" information depends on the method of authentication that was used to perform the control plane operation. If the authorization was performed by an Azure AD security principal, the object identifier of that security principal would also appear in this JSON output (For example:

```
"http://schemas.microsoft.com/identity/claims/objectidentifier": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx").
```

Because you might not always see other identity-related information such as an email address or name, the object identifier is always the best way to uniquely identify the security principal.

You can find the friendly name of that security principal by taking the value of the object identifier, and searching for the security principal in Azure AD page of the Azure portal. The following screenshot shows a search result in Azure AD.

The screenshot shows the Microsoft Azure (Preview) portal with the Azure Active Directory Overview page. The search bar at the top contains 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'. The 'Enterprise applications' section is highlighted with a red box, showing the entry 'contoso-service-principal'.

## Auditing data plane operations

Data plane operations are captured in [Azure resource logs for Storage](#). You can [configure Diagnostic setting](#) to export logs to Log Analytics workspace for a native query experience.

Here's a Log Analytics query that retrieves the "when", "who", "what", and "how" information in a list of log entries.

```
StorageBlobLogs
| where TimeGenerated > ago(3d)
| project TimeGenerated, AuthenticationType, RequesterObjectId, OperationName, Uri
```

For the "when" portion of your audit, the `TimeGenerated` field shows when the log entry was recorded.

For the "what" portion of your audit, the `Uri` field shows the item was modified or read.

For the "how" portion of your audit, the `OperationName` field shows which operation was executed.

For the "who" portion of your audit, `AuthenticationType` shows which type of authentication was used to make a request. This field can show any of the types of authentication that Azure Storage supports including the use of an account key, a SAS token, or Azure Active Directory (Azure AD) authentication.

### Identifying the security principal used to authorize a request

If a request was authenticated by using Azure AD, the `RequesterObjectId` field provides the most reliable way to identify the security principal. You can find the friendly name of that security principal by taking the value of the `RequesterObjectId` field, and searching for the security principal in Azure AD page of the Azure portal. The following screenshot shows a search result in Azure AD.

The screenshot shows the Microsoft Azure (Preview) portal with the Azure Active Directory Overview page selected. The search bar at the top contains 'Search resources, services, and docs (G+ /)'. The main content area has tabs for 'Overview', 'Monitoring', and 'Tutorials'. A search bar in the center contains the text 'xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxx'. Below it, sections for 'Users', 'Devices', 'Enterprise applications', 'Groups', 'App registrations', 'Roles', and 'Administrative units' show 'No results.' except for 'Enterprise applications' which lists 'contoso-service-principal'.

In some cases, a user principal name or *UPN* might appear in logs. For example, if the security principal is an Azure AD user, the UPN will likely appear. For other types of security principals such as user assigned managed identities, or in certain scenarios such as cross Azure AD tenant authentication, the UPN will not appear in logs.

This query shows all read operations performed by OAuth security principals.

```
StorageBlobLogs
| where TimeGenerated > ago(3d)
and OperationName == "GetBlob"
and AuthenticationType == "OAuth"
| project TimeGenerated, AuthenticationType, RequesterObjectId, OperationName, Uri
```

Shared Key and SAS authentication provide no means of auditing individual identities. Therefore, if you want to improve your ability to audit based on identity, we recommend that you transition to Azure AD, and prevent shared key and SAS authentication. To learn how to prevent Shared Key and SAS authentication, see [Prevent Shared Key authorization for an Azure Storage account](#). To get started with Azure AD, see [Authorize access to blobs using Azure Active Directory](#).

#### Identifying the SAS token used to authorize a request

You can query for operations that were authorized by using a SAS token. For example, this query returns all write operations that were authorized by using a SAS token.

```
StorageBlobLogs
| where TimeGenerated > ago(3d)
and OperationName == "PutBlob"
and AuthenticationType == "SAS"
| project TimeGenerated, AuthenticationType, AuthenticationHash, OperationName, Uri
```

For security reasons, SAS tokens don't appear in logs. However, the SHA-256 hash of the SAS token will appear in the `AuthenticationHash` field that is returned by this query.

If you've distributed several SAS tokens, and you want to know which SAS tokens are being used, you'll have to convert each of your SAS tokens to a SHA-256 hash, and then compare that hash to the hash value that appears in logs.

First decode each SAS token string. The following example decodes a SAS token string by using PowerShell.

```
[uri]::UnescapeDataString("<SAS token goes here>")
```

Then, you can pass that string to the [Get-FileHash](#) PowerShell cmdlet. For an example, see [Example 4: Compute the hash of a string](#).

Alternatively, you can pass the decoded string to the [hash\\_sha256\(\)](#) function as part of a query when you use Azure Data Explorer.

SAS tokens do not contain identity information. One way to track the activities of users or organizations, is to keep a mapping of users or organizations to various SAS token hashes.

## Optimize cost for infrequent queries

You can export logs to Log Analytics for rich native query capabilities. When you have massive transactions on your storage account, the cost of using logs with Log Analytics might be high. For more information, see [Azure Log Analytics Pricing](#). If you only plan to query logs occasionally (for example, query logs for compliance auditing), you can consider reducing the total cost by exporting logs to storage account, and then using a serverless query solution on top of log data, for example, Azure Synapse.

With Azure Synapse, you can create server-less SQL pool to query log data when you need. This could save costs significantly.

1. Export logs to storage account. For more information, see [Creating a diagnostic setting](#).
2. Create and configure a Synapse workspace. For more information, see [Quickstart: Create a Synapse workspace](#).
3. Query logs. For more information, see [Query JSON files using serverless SQL pool in Azure Synapse Analytics](#).

Here's an example:

```
select
 JSON_VALUE(doc, '$.time') AS time,
 JSON_VALUE(doc, '$.properties.accountName') AS accountName,
 JSON_VALUE(doc, '$.identity.type') AS identityType,
 JSON_VALUE(doc, '$.identity.requester.objectId') AS requesterObjectId,
 JSON_VALUE(doc, '$.operationName') AS operationName,
 JSON_VALUE(doc, '$.callerIpAddress') AS callerIpAddress,
 JSON_VALUE(doc, '$.uri') AS uri
 doc
from openrowset(
 bulk 'https://demo2uswest4log.blob.core.windows.net/insights-logs-
storageread/resourceId=/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxx/resourceGroups/mytestrp/providers/Microsoft.Storage/storageAccounts/demo2uswest/blobServ-
ices/default/y=2021/m=03/d=19/h=*/m=*/PT1H.json',
 format = 'csv', fieldterminator = '0x0b', fieldquote = '0x0b'
) with (doc nvarchar(max)) as rows
order by JSON_VALUE(doc, '$.time') desc
```

## See also

- [Monitoring Azure Blob Storage](#).
- [Azure Blob Storage monitoring data reference](#)
- [Tutorial: Use Kusto queries in Azure Data Explorer and Azure Monitor](#).
- [Get started with log queries in Azure Monitor](#).

# Monitoring your storage service with Azure Monitor Storage insights

8/22/2022 • 12 minutes to read • [Edit Online](#)

Storage insights provides comprehensive monitoring of your Azure Storage accounts by delivering a unified view of your Azure Storage services performance, capacity, and availability. You can observe storage capacity, and performance in two ways, view directly from a storage account or view from Azure Monitor to see across groups of storage accounts.

This article will help you understand the experience Storage insights delivers to derive actionable knowledge on the health and performance of Storage accounts at scale, with a capability to focus on hotspots and diagnose latency, throttling, and availability issues.

## Introduction to Storage insights

Before diving into the experience, you should understand how it presents and visualizes information. Whether you select the Storage feature directly from a storage account or from Azure Monitor, Storage insights presents a consistent experience.

Combined it delivers:

- **At scale perspective** showing a snapshot view of their availability based on the health of the storage service or the API operation, utilization showing total number of requests that the storage service receives, and latency showing the average time the storage service or API operation type is taking to process requests. You can also view capacity by blob, file, table, and queue.
- **Drill down analysis** of a particular storage account to help diagnose issues or perform detailed analysis by category – availability, performance, failures, and capacity. Selecting any one of those options provides an in-depth view of metrics.
- **Customizable** where you can change which metrics you want to see, modify or set thresholds that align with your limits, and save as your own workbook. Charts in the workbook can be pinned to Azure dashboard.

This feature does not require you to enable or configure anything, the storage metrics from your storage accounts are collected by default. If you are unfamiliar with metrics available on Azure Storage, view the description and definition in Azure Storage metrics by reviewing [Azure storage metrics](#).

### NOTE

There is no charge to access this feature and you will only be charged for the Azure Monitor essential features you configure or enable, as described on the [Azure Monitor pricing details](#) page.

## View from Azure Monitor

From Azure Monitor, you can view transaction, latency, and capacity details from multiple storage accounts in your subscription, and help identify performance, capacity problems, and failures.

To view the utilization and availability of your storage accounts across all of your subscriptions, perform the following steps.

1. Sign in to the Azure portal.
2. Select **Monitor** from the left-hand pane in the Azure portal, and under the **Insights** section, select **Storage Accounts**.

Subscription	Transactions	Transactions Timeline	Availability	E2E Latency	Server Latency	Client/Other Error/Errors
CloudOps Tip1	130.6K	<div style="width: 100%;"> </div>	100%	61.51ms	45.25ms	24.4K
a64sl75plosp2diag0	48.3K	<div style="width: 100%;"> </div>	100%	5.77ms	5.19ms	16
a64sl75plosp2agntpub	26.9K	<div style="width: 100%;"> </div>	100%	5.58ms	5.48ms	13
00a64sl75plosp2agntpri0	26.2K	<div style="width: 100%;"> </div>	100%	7.56ms	6.82ms	
a64sl75plosp2mstr0	22.6K	<div style="width: 100%;"> </div>	100%	11.47ms	2.84ms	
60ybtm66zbygrzeswarm2	3.3K	<div style="width: 100%;"> </div>	100%	12.04ms	3.59ms	
00ybtm66zbygrzeswarm1	3.3K	<div style="width: 100%;"> </div>	100%	20.21ms	20.08ms	
00kwwjhgymkdx5gagn0	24	<div style="width: 100%;"> </div>	100%	19.21ms	19.04ms	
60a64sl75plosp2agntpri1	24	<div style="width: 100%;"> </div>	100%	12.08ms	11.71ms	
691965eastus	24	<div style="width: 100%;"> </div>	100%	13.17ms	12.88ms	
a64sl75plosp2exhb0	24	<div style="width: 100%;"> </div>	100%			

## Overview workbook

On the **Overview** workbook for the selected subscription, the table displays interactive storage metrics and service availability state for up to 5 storage accounts grouped within the subscription. You can filter the results based on the options you select from the following drop-down lists:

- **Subscriptions** - only subscriptions that have storage accounts are listed.
- **Storage Accounts** - by default, 5 storage accounts are pre-selected. If you select all or multiple storage accounts in the scope selector, up to 200 storage accounts will be returned. For example, if you had a total of 573 storage accounts across three subscriptions that you've selected, only 200 accounts would be displayed.
- **Time Range** - by default, displays the last 4 hours of information based on the corresponding selections made.

The counter tile under the drop-down lists rolls-up the total number of storage accounts in the subscription and reflects how many of the total are selected. There is conditional color-coding or heatmaps for columns in the workbook that report transaction metrics or errors. The deepest color has the highest value and a lighter color is based on the lowest values. For the error-based columns, the value is in red and for the metric-based columns, the value is in blue.

Select a value in the columns **Availability**, **E2E Latency**, **Server Latency**, and **transaction error type/Errors** directs you to a report tailored to the specific type of storage metrics that match the column selected for that storage account. For more information about the workbooks for each category, see the [Detailed storage workbooks](#) section below.

### NOTE

For details on which errors can be shown in the report, see [Response Type schema](#) and look for response types such as **ServerError**, **ClientError**, **ClientThrottlingError**. Depending on the storage accounts selected, if there are more than three types of errors reported, all other errors are represented under the category of **Other**.

The default **Availability** threshold is:

- Warning - 99%
- Critical - 90%

To set an availability threshold based on the results of your observation or requirements, review [modify the availability threshold](#).

## Capacity workbook

Select **Capacity** at the top of the page and the **Capacity** workbook opens. It shows you the amount of total storage used in the account and capacity used by each data service in the account to help identify over and under utilized storage.

The screenshot shows the Azure Capacity workbook interface. At the top, there are navigation links: Home > Monitor - Storage Accounts (preview) > Capacity. Below this is a toolbar with icons for Gallery, Edit, Refresh, and others. The main area has three dropdown menus: Subscriptions (CloudOps Tip1), Storage Accounts (All), and Time Range (Last 4 hours). A large bold number '79 / 79' is displayed. Below it, a section titled 'Storage accounts' has tabs for Overview, Capacity (which is selected and highlighted in blue), and Learn more. A search bar is present. The main content area is a table with columns: Subscription, Account used capacity, Account used capacity Timeline, Blob capacity, File capacity, Queue capacity, and Table capacity. The table lists 10 storage accounts under the 'CloudOps Tip1' subscription. The 'Account used capacity' column uses conditional color-coding, with most values being blue (highest value) and one being white (lowest value). The 'Table capacity' column also uses this color-coding. The table is scrollable, indicated by a vertical scrollbar on the right.

Subscription	Account used capacity	Account used capacity Timeline	Blob capacity	File capacity	Queue capacity	Table capacity
CloudOps Tip1	3TiB		98.1MiB	0B	0B	3TiB
a64sl75plosp2diag0	50GiB		47.5KiB	50GiB	0B	2.6MiB
cs4692aea0b2d89x4e7xae3	32.6GiB		32.6GiB	0B	0B	4.6MiB
ybtm6zbygrzeswarm0	30.5GiB		30.4GiB	0B	0B	142.7MiB
sfflogshicidemo3330	29.4GiB		29.3GiB	0B	0B	4.2MiB
00a64sl75plosp2agntrp0	29.2GiB		29.2GiB	0B	0B	4MiB
00ybtm6zbygrzeswarm1	28.7GiB		28.7GiB	0B	0B	0B
kwwjhgymld5gmstr0	19.5GiB		19.5GiB	0B	0B	2.5MiB
00kwjhgymld5gagnrt0	16.7GiB		16.7GiB	0B	0B	2.5MiB
a64sl75plosp2agnpub	11.6GiB		11.6GiB	0B	0B	4MiB

There is conditional color-coding or heatmaps for columns in the workbook that report capacity metrics with a blue value. The deepest color has the highest value and a lighter color is based on the lowest values.

When you select a value under any one of the columns in the workbook, you drill down to the **Capacity** workbook for the storage account. Further details about the drill-down report are described in the [Detailed storage workbooks](#) section below.

## View from a storage account

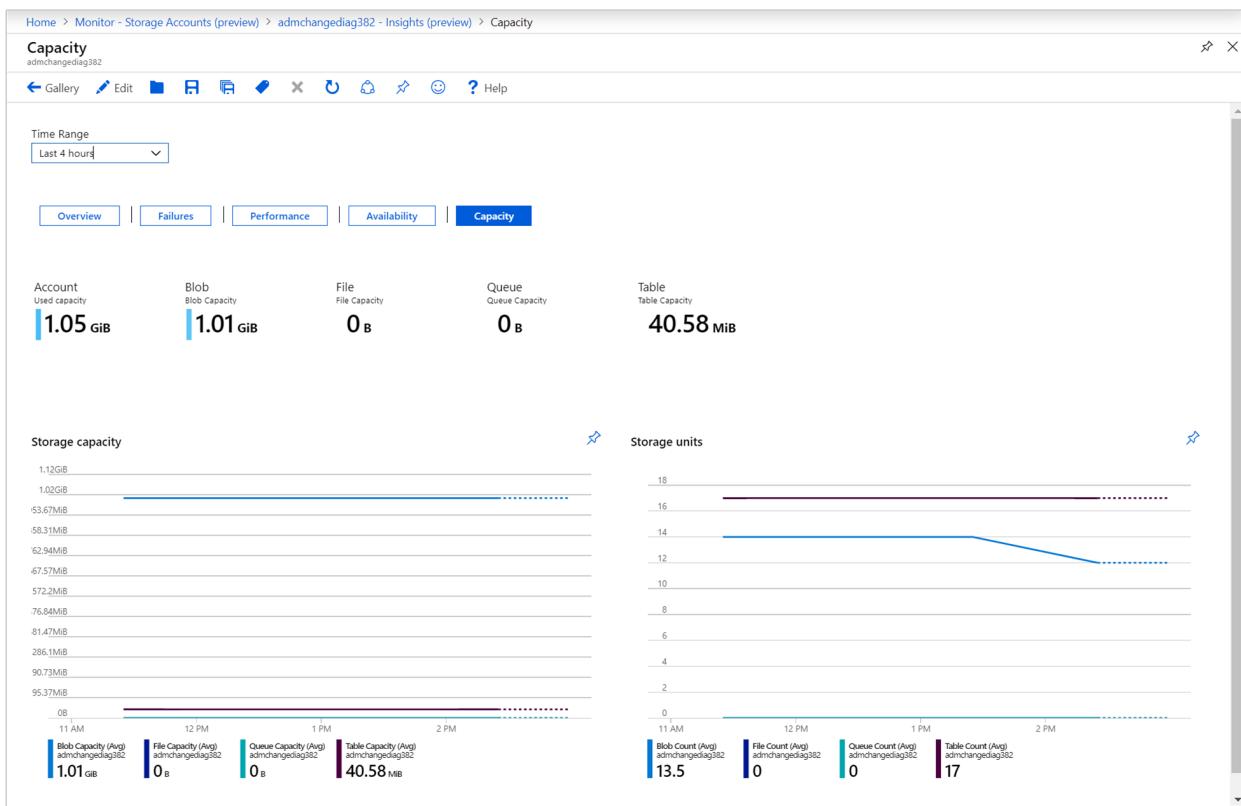
To access VM insights directly from a storage account:

1. In the Azure portal, select Storage accounts.
2. From the list, choose a storage account. In the Monitoring section, choose Insights.

On the **Overview** workbook for the storage account, it shows several storage performance metrics that help you quickly assess:

- Health of the Storage service to immediately see if an issue outside of your control is affecting the Storage service in the region it is deployed to, which is stated under the **Summary** column.
- Interactive performance charts showing the most essential details related to storage capacity, availability, transactions, and latency.
- Metric and status tiles highlighting service availability, total count of transactions to the storage service, E2E latency, and server latency.

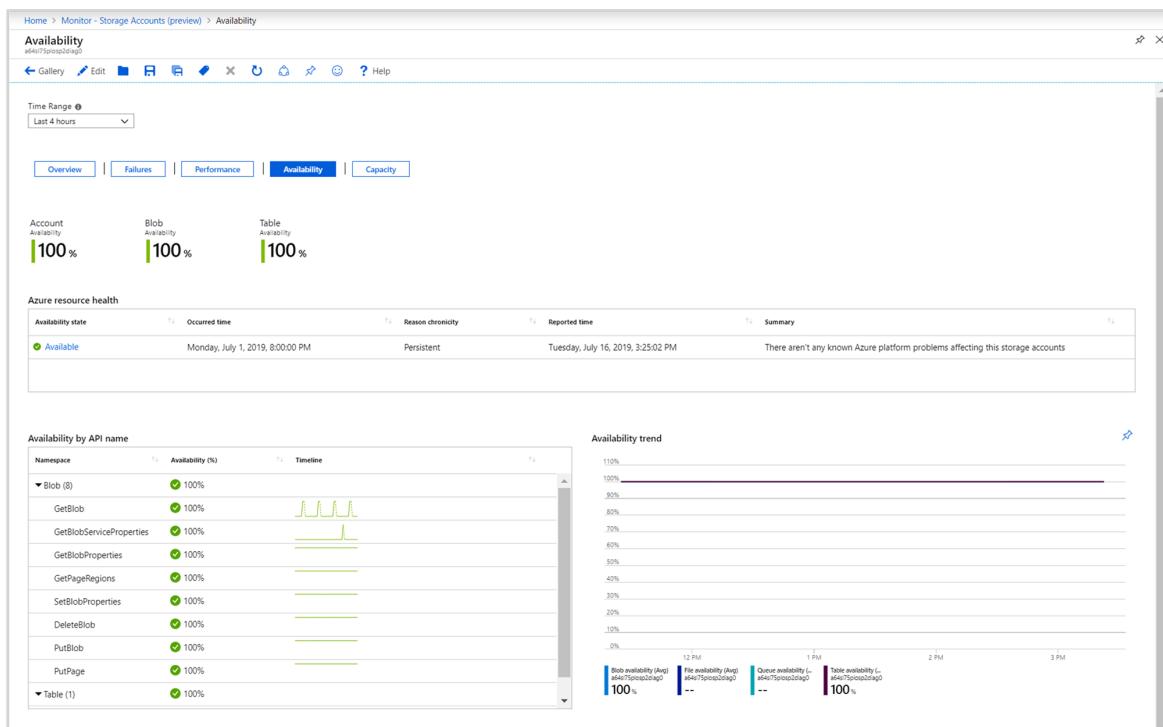
Selecting any one of buttons for **Failures**, **Performance**, **Availability**, and **Capacity** opens the respective workbook.



## Detailed storage workbooks

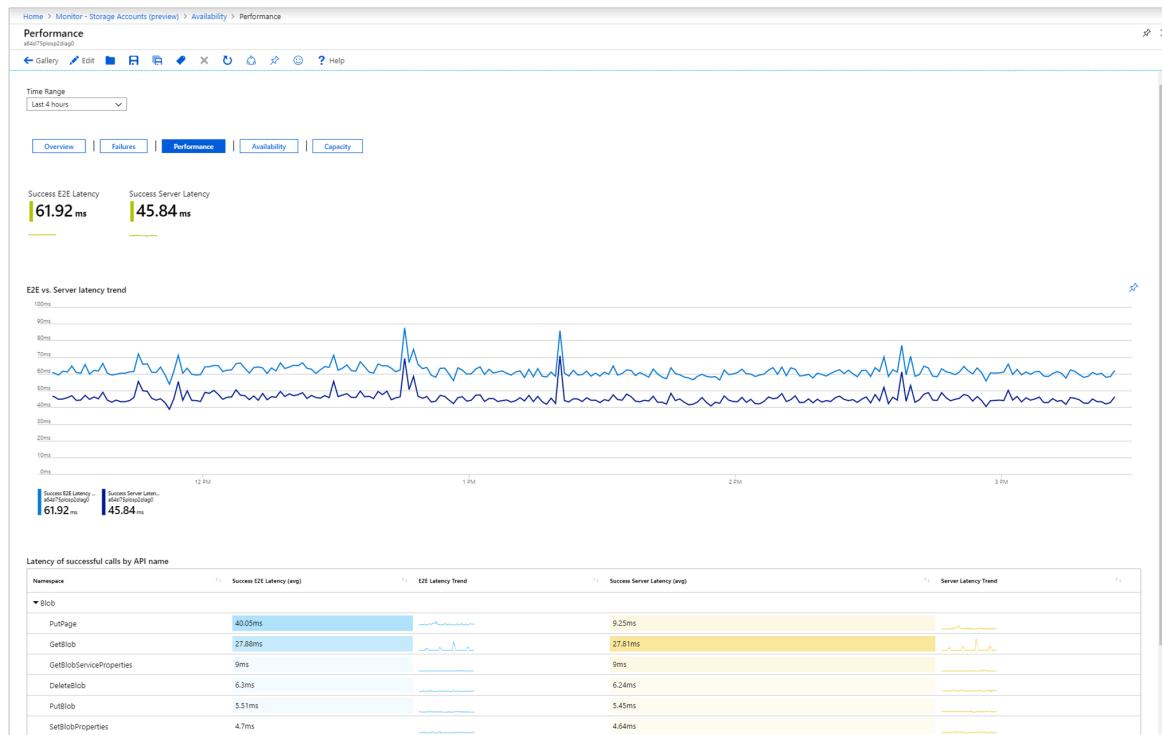
Whether you selected a value in the columns **Availability**, **E2E Latency**, **Server Latency**, and **transaction error type/Errors** from the multiple storage account **Overview** workbook, or selecting any one of buttons for **Failures**, **Performance**, **Availability**, and **Capacity** from the **Overview** workbook from a specific storage account, each deliver a set of interactive storage-related information tailored to that category.

- **Availability** opens the **Availability** workbook. It shows the current health state of Azure Storage service, a table showing the available health state of each object categorized by data service defined in the storage account with a trend line representing the time range selected, and an availability trend chart for each data service in the account.

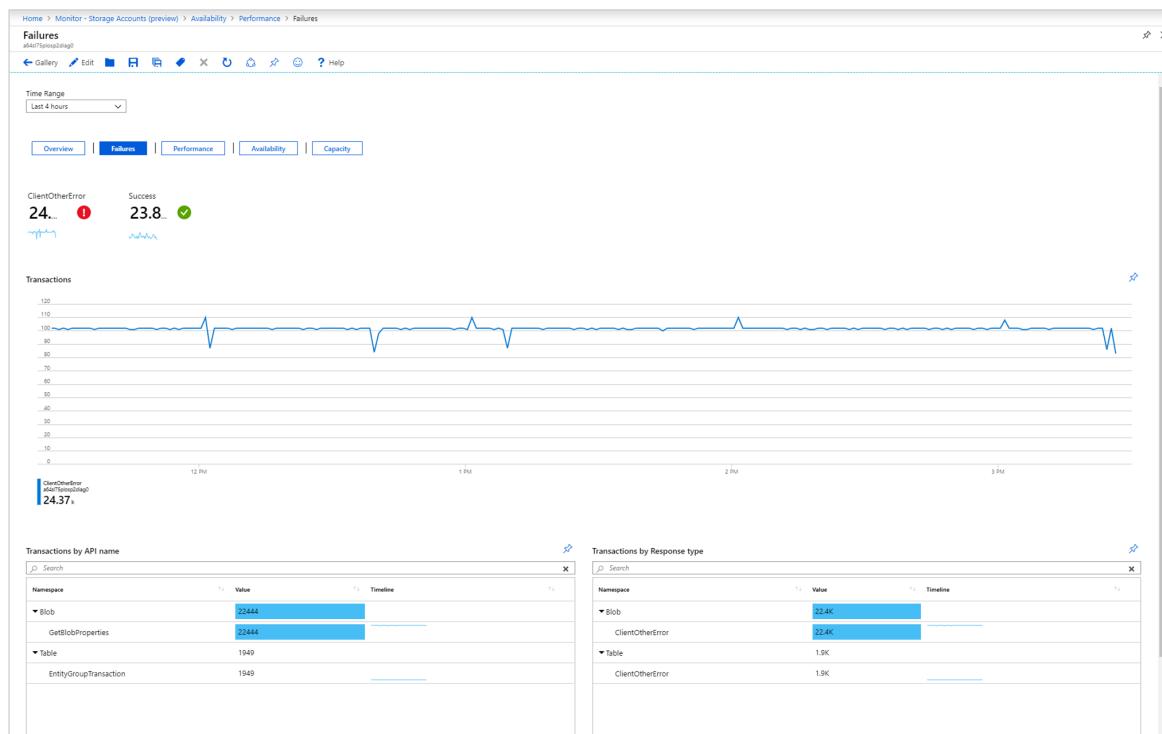


- **E2E Latency** and **Server Latency** opens the **Performance** workbook. It includes a rollup status tile

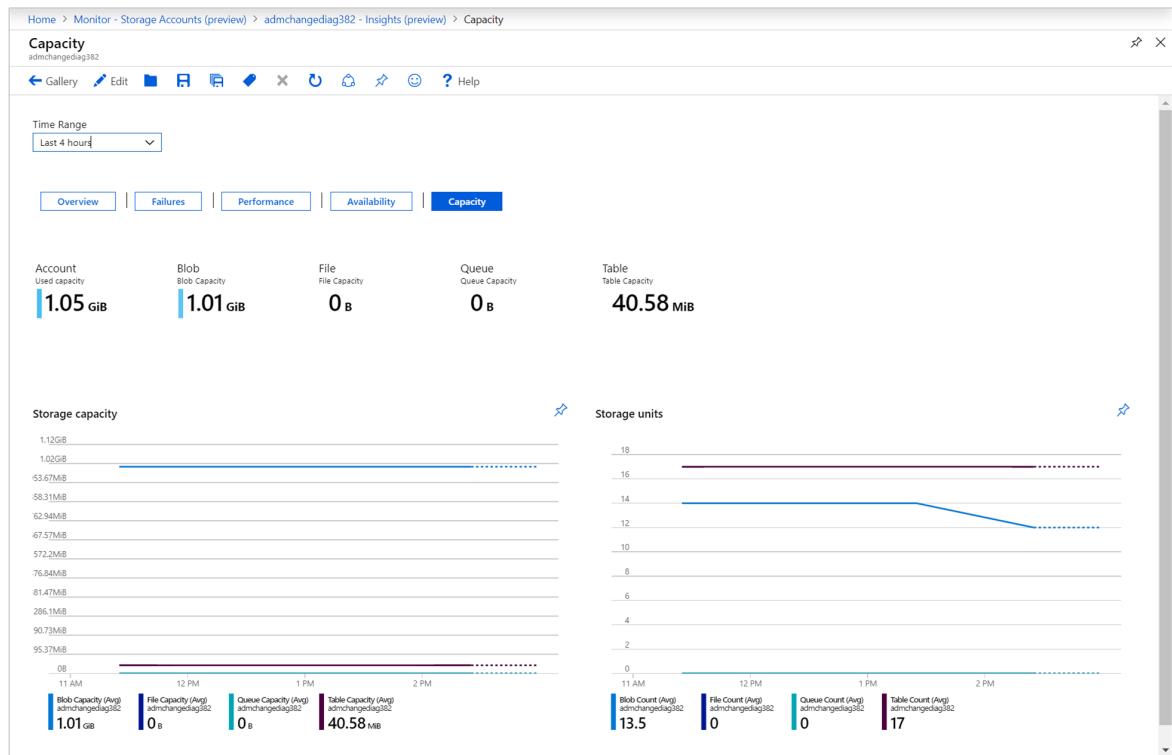
showing E2E latency and server latency, a performance chart of E2E versus server latency, and a table breaking down latency of successful calls by API categorized by data service defined in the storage account.



- Selecting any of the error categories listed in the grid open the **Failure** workbook. The report shows metric tiles of all other client-side errors except described ones and successful requests, client-throttling errors, a performance chart for the transaction **Response Type** dimension metric specific to ClientOtherError attribute, and two tables - **Transactions by API name** and **Transactions by Response type**.



- Capacity** opens the **Capacity** workbook. It shows the total amount of storage used for each storage data object in the account in the tiles and the chart, and how many data objects are stored in the account.



## Pin and export

You can pin any one of the metric sections to an Azure Dashboard by selecting the pushpin icon at the top right of the section.

This screenshot shows the same dashboard as above, but with a specific section pinned to the top right. A red box highlights the pushpin icon with a downward arrow, indicating where to click to pin the section. Below the pinned section, there is a search bar and a table titled 'Subscription' with columns for Account used capacity, Account used capacity Timeline, Blob capacity, File capacity, Queue capacity, and Table capacity. The table lists various storage accounts with their respective capacity values.

Subscription	Account used capacity	Account used capacity Timeline	Blob capacity	File capacity	Queue capacity	Table capacity
Microsoft Azure						
cs268627f8c65b8x4601xb48	5GiB		0B	5GiB	0B	2.6MiB
cs768627f8c65b8x4601xb48	5GiB		0B	5GiB	0B	2.4MiB
infralabdiag	5GiB		4.9MiB	0B	0B	5GiB
infralabdiag106	3.8GiB		627.2KiB	0B	0B	3.8GiB
infralabdiag774	44.7MiB		0B	0B	0B	44.7MiB
azureautoacomsstorage	8.8MiB		6.3MiB	0B	0B	2.5MiB
infralabdiag553	4.7MiB		0B	0B	0B	4.7MiB
awsautomationomsstorage	3.3MiB		805.2KiB	0B	0B	2.5MiB
infralabdiag712	2.9MiB		0B	0B	0B	2.9MiB
infralabdiag205	2.6MiB		0B	0B	0B	2.6MiB

The multi-subscription and storage account **Overview** or **Capacity** workbooks support exporting the results in Excel format by selecting the down arrow icon to the right of the pushpin icon.

The screenshot shows a table with columns for Subscription, Account used capacity, Account used capacity Timeline, Blob capacity, File capacity, Queue capacity, and Table capacity. A search bar at the top left and a download icon at the top right are also visible.

Subscription	Account used capacity	Account used capacity Timeline	Blob capacity	File capacity	Queue capacity	Table capacity
<b>Microsoft Azure</b>						
cs268627f8c65b8x4601xb48	5GiB		0B	5GiB	0B	2.6MiB
cs768627f8c65b8x4601xb48	5GiB		0B	5GiB	0B	2.4MiB
infralabdiag	5GiB		4.9MiB	0B	0B	5GiB
infralabdiag106	3.8GiB		627.2KiB	0B	0B	3.8GiB
infralabdiag774	44.7MiB		0B	0B	0B	44.7MiB
azureautoacccomsstorage	8.8MiB		6.3MiB	0B	0B	2.5MiB
infralabdiag553	4.7MiB		0B	0B	0B	4.7MiB
awsautomationomsstorage	3.3MiB		805.2KiB	0B	0B	2.5MiB
infralabdiag712	2.9MiB		0B	0B	0B	2.9MiB
infralabdiag205	2.6MiB		0B	0B	0B	2.6MiB

## Customize Storage insights

This section highlights common scenarios for editing the workbook to customize in support of your data analytics needs:

- Scope the workbook to always select a particular subscription or storage account(s)
- Change metrics in the grid
- Change the availability threshold
- Change the color rendering

The customizations are saved to a custom workbook to prevent overwriting the default configuration in our published workbook. Workbooks are saved within a resource group, either in the **My Reports** section that's private to you or in the **Shared Reports** section that's accessible to everyone with access to the resource group. After you save the custom workbook, you need to go to the workbook gallery to launch it.

The screenshot shows the command bar of the workbook with options like Search, Gallery, Edit, and Save.

### Specifying a subscription or storage account

You can configure the multi-subscription and storage account **Overview** or **Capacity** workbooks to scope to a particular subscription(s) or storage account(s) on every run, perform the following steps.

1. Select **Monitor** from the portal and then select **Storage Accounts** from the left-hand pane.
2. On the **Overview** workbook, from the command bar select **Edit**.
3. Select from the **Subscriptions** drop-down list one or more subscriptions you want it to default to.  
Remember, the workbook supports selecting up to a total of 10 subscriptions.
4. Select from the **Storage Accounts** drop-down list one or more accounts you want it to default to.  
Remember, the workbook supports selecting up to a total of 200 storage accounts.
5. Select **Save as** from the command bar to save a copy of the workbook with your customizations, and then click **Done editing** to return to reading mode.

### Modify metrics and colors in the workbook

The prebuilt workbooks contain metric data and you have the ability to modify or remove any one of the visualizations and customize to your team's specific needs.

In our example, we are working with the multi-subscription and storage account capacity workbook, to

demonstrate how to:

- Remove a metric
- Change color rendering

You can perform the same changes against any one of the prebuilt **Failures**, **Performance**, **Availability**, and **Capacity** workbooks.

1. Select **Monitor** from the portal and then select **Storage Accounts** from the left-hand pane.
2. Select **Capacity** to switch to the capacity workbook and from the command bar, select **Edit** from the command bar.

The screenshot shows the Azure Monitor interface for Storage Accounts. At the top, there's a navigation bar with 'Gallery' and a red-highlighted 'Edit' button. Below it are three dropdown menus: 'Subscriptions' set to 'Microsoft Azure', 'Storage Accounts' set to '10 selected', and 'Time Range' set to 'Last 4 hours'. The main area displays a table of storage account metrics.

3. Next to the metrics section, select **Edit**.

The screenshot shows the 'Capacity' tab of the metrics grid. It includes a search bar, a toolbar with 'Edit' and other options, and a table of storage account metrics. The table has columns for Subscription, Account used capacity, Account used capacity Timeline, Blob capacity, File capacity, Queue capacity, and Table capacity. The 'Edit' button is highlighted at the bottom right of the grid.

4. We are going to remove the **Account used capacity timeline** column, so select **Column Settings** in the metrics grid.

The screenshot shows the 'Column Settings' pane for the metrics grid. It includes a toolbar with filters for Resource, Time Range, Visualization, Format, and Size, and buttons for Used capacity, Average, Blob Capacity, Average, File Capacity, Average, Queue Capacity, Average, Table Capacity, Average, Add metric, and Add filter. Below is a table of storage account metrics with the 'Account used capacity Timeline' column removed. The 'Edit' button is highlighted at the bottom right of the grid.

5. In the **Edit column settings** pane, select under the **Columns** section  
`microsoft.storage/storageaccounts-Capacity-UsedCapacity Timeline$|Account used capacity`

Timeline\$, and under the drop-down list Column renderer select Hidden.

6. Select Save and close to commit your change.

Now let's change the color theme for the capacity metrics in the report to use green instead of blue.

1. Select Column Settings in the metrics grid.
2. In the Edit column settings pane, select under the Columns section  
microsoft.storage/storageaccounts-Capacity-UsedCapacity\$ |  
microsoft.storage/storageaccounts/blobservices-Capacity-BlobCapacity\$ |  
microsoft.storage/storageaccounts/fileservices-Capacity-FileCapacity\$ |  
microsoft.storage/storageaccounts/queueservices-Capacity-QueueCapacity\$ |  
microsoft.storage/storageaccounts/tableservices-Capacity-TableCapacity\$. Under the drop-down list Color palette, select Green.
3. Select Save and close to commit your change.
4. Select Save as from the command bar to save a copy of the workbook with your customizations, and then click Done editing to return to reading mode.

### Modify the availability threshold

In this example, we are working with the storage account capacity workbook and demonstrating how to modify the availability threshold. By default, the tile and grid reporting percent availability are configured with a minimum threshold of 90 and maximum threshold of 99. We are going to change the minimum threshold value of the Availability % in the Availability by API name grid to 85%, which means the health state changes to critical if the threshold is less than 85 percent.

1. Select Storage accounts from the portal and then select a storage account from the list.
2. Select Insights from the left-hand pane.
3. In the workbook, select Availability to switch to the availability workbook, and then select Edit from the command bar.
4. Scroll down to the bottom of the page and on the left-hand side next to the Availability by API grid, select Edit.

Availability by API name			
Namespace	↑↓	Availability (%)	↑↓ Timeline
▼ Table (1)		✓ 100%	
EntityGroupTransacti...		✓ 100%	—

1 Edit

5. Select **Column settings** and then in the **Edit column settings** pane, under the **Columns** section select **Availability (%) (Thresholds + Formatted)**.
6. Change the value for the **Critical** health state from **90** to **85** and then click **Save and Close**.

7. Select **Save as** from the command bar to save a copy of the workbook with your customizations, and then click **Done editing** to return to reading mode.

## Troubleshooting

For general troubleshooting guidance, refer to the dedicated workbook-based insights [troubleshooting article](#).

This section will help you with the diagnosis and troubleshooting of some of the common issues you may encounter when using Storage insights. Use the list below to locate the information relevant to your specific issue.

## Resolving performance, capacity, or availability issues

To help troubleshoot any storage-related issues you identify with Storage insights, see the Azure Storage troubleshooting guidance.

### Why can I only see 200 storage accounts?

The number of selected storage accounts has a limit of 200, regardless of the number of subscriptions that are selected.

### How to change the coloring and threshold for availability?

Refer to the [Modify the availability threshold](#) section for the detailed steps on how to change the coloring and thresholds for availability.

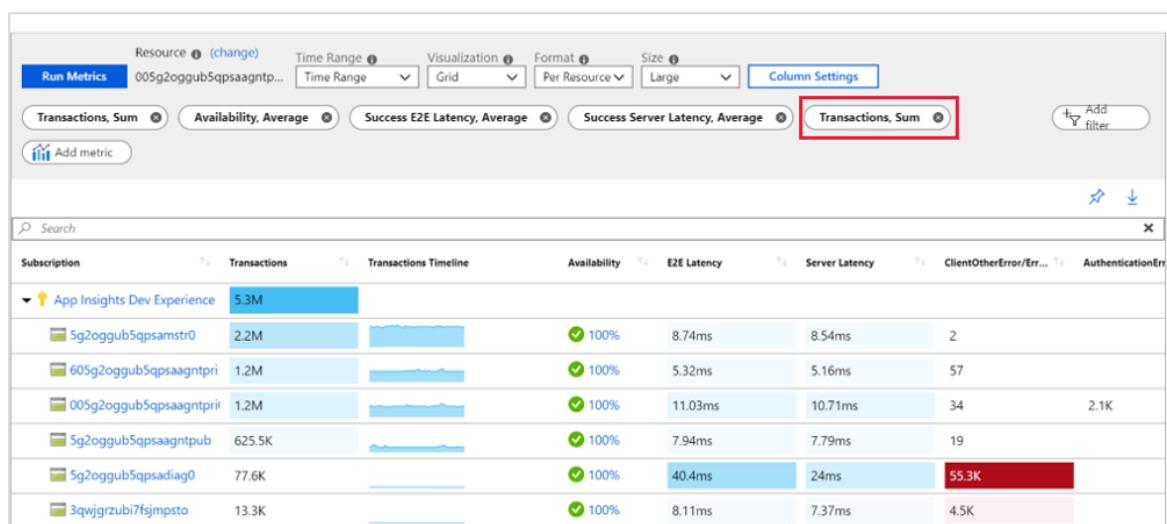
### How to analyze and troubleshoot the data shown in Storage insights?

Refer to the [Monitor, diagnose, and troubleshoot Microsoft Azure Storage](#) article for details on how to analyze and troubleshoot the Azure Storage data shown in Storage insights.

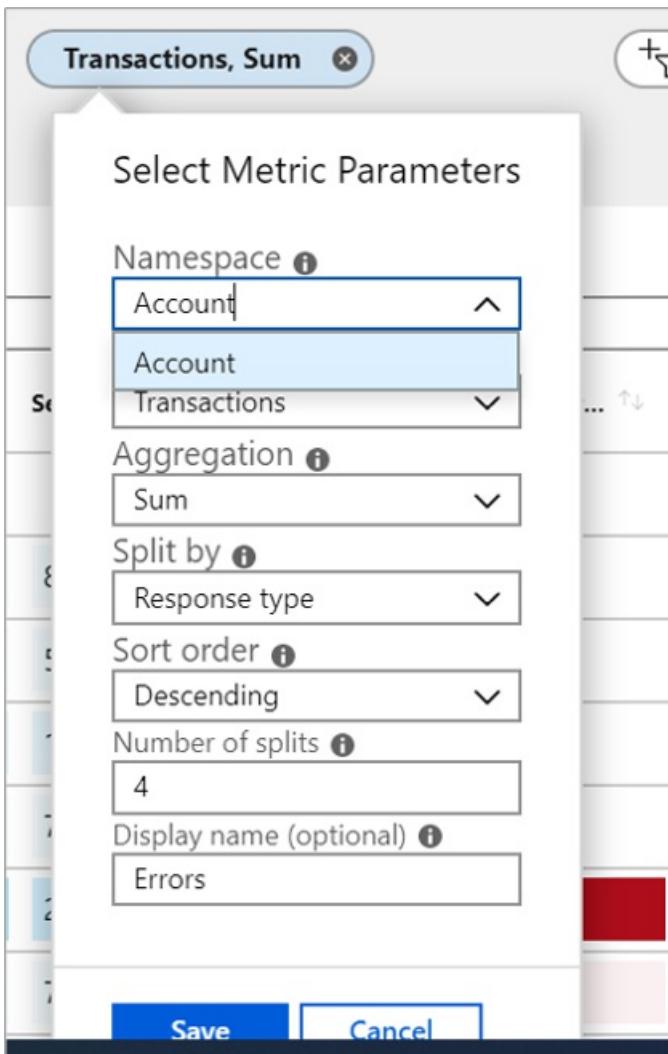
### Why don't I see all the types of errors in metrics?

Currently, up to three different types of errors are shown and the rest of the errors are grouped together in a single bucket. It is controlled using splitByLimit and can be modified. For changing this property:

1. Click on edit workbook.
2. Go to metrics, click on edit and then select **Transactions, Sum** or whatever metrics you want to edit.



3. Then change the Number of Splits.



If you want to see n different types of error than specify splitByLimit as n+1, 1 extra for rest of the errors.

#### I saved my workbook while on some Storage Account. Why can't I find it now?

Each workbook is saved in the storage account that you saved it in. Try to find the specific Storage Account in which the user saved the workbook. Otherwise, there is no way to find a specific workbook without knowing the resource (storage account).

## Next steps

- Configure [metric alerts](#) and [service health notifications](#) to set up automated alerting to aid in detecting issues.
- Learn the scenarios workbooks are designed to support, how to author new and customize existing reports, and more by reviewing [Create interactive reports with Azure Monitor workbooks](#).
- For an in-depth guide on using Storage Analytics and other tools to identify, diagnose, and troubleshoot Azure Storage-related issues, see [Monitor, diagnose, and troubleshoot Microsoft Azure Storage](#).

# Troubleshoot performance in Azure Storage accounts

8/22/2022 • 7 minutes to read • [Edit Online](#)

This article helps you investigate unexpected changes in behavior (such as slower than usual response times). These changes in behavior can often be identified by monitoring storage metrics in Azure Monitor. For general information about using metrics and logs in Azure Monitor, see

- [Monitoring Azure Blob Storage](#)
- [Monitoring Azure Files](#)
- [Monitoring Azure Queue Storage](#)
- [Monitoring Azure Table storage](#)

## Monitoring performance

To monitor the performance of the storage services, you can use the following metrics.

- The **SuccessE2ELatency** and **SuccessServerLatency** metrics show the average time the storage service or API operation type is taking to process requests. **SuccessE2ELatency** is a measure of end-to-end latency that includes the time taken to read the request and send the response in addition to the time taken to process the request (therefore includes network latency once the request reaches the storage service); **SuccessServerLatency** is a measure of just the processing time and therefore excludes any network latency related to communicating with the client.
- The **Egress** and **Ingress** metrics show the total amount of data, in bytes, coming in to and going out of your storage service or through a specific API operation type.
- The **Transactions** metric shows the total number of requests that the storage service or API operation is receiving. **Transactions** is the total number of requests that the storage service receives.

You can monitor for unexpected changes in any of these values. These changes could indicate an issue that requires further investigation.

In the [Azure portal](#), you can add alert rules which notify you when any of the performance metrics for this service fall below or exceed a threshold that you specify.

## Diagnose performance issues

The performance of an application can be subjective, especially from a user perspective. Therefore, it is important to have baseline metrics available to help you identify where there might be a performance issue. Many factors might affect the performance of an Azure storage service from the client application perspective. These factors might operate in the storage service, in the client, or in the network infrastructure; therefore it is important to have a strategy for identifying the origin of the performance issue.

After you have identified the likely location of the cause of the performance issue from the metrics, you can then use the log files to find detailed information to diagnose and troubleshoot the problem further.

## Metrics show high SuccessE2ELatency and low SuccessServerLatency

In some cases, you might find that **SuccessE2ELatency** is significantly higher than the **SuccessServerLatency**. The storage service only calculates the metric **SuccessE2ELatency** for successful

requests and, unlike **SuccessServerLatency**, includes the time the client takes to send the data and receive acknowledgment from the storage service. Therefore, a difference between **SuccessE2ELatency** and **SuccessServerLatency** could be either due to the client application being slow to respond, or due to conditions on the network.

**NOTE**

You can also view **E2ELatency** and **ServerLatency** for individual storage operations in the Storage Logging log data.

### Investigating client performance issues

Possible reasons for the client responding slowly include having a limited number of available connections or threads, or being low on resources such as CPU, memory or network bandwidth. You may be able to resolve the issue by modifying the client code to be more efficient (for example by using asynchronous calls to the storage service), or by using a larger Virtual Machine (with more cores and more memory).

For the table and queue services, the Nagle algorithm can also cause high **SuccessE2ELatency** as compared to **SuccessServerLatency**: for more information, see the post [Nagle's Algorithm is Not Friendly towards Small Requests](#). You can disable the Nagle algorithm in code by using the **ServicePointManager** class in the **System.Net** namespace. You should do this before you make any calls to the table or queue services in your application since this does not affect connections that are already open. The following example comes from the **Application\_Start** method in a worker role.

```
var connectionString = Constants.connectionString;

QueueServiceClient queueServiceClient = new QueueServiceClient(connectionString);

ServicePoint queueServicePoint = ServicePointManager.FindServicePoint(queueServiceClient.Uri);
queueServicePoint.UseNagleAlgorithm = false;
```

You should check the client-side logs to see how many requests your client application is submitting, and check for general .NET related performance bottlenecks in your client such as CPU, .NET garbage collection, network utilization, or memory. As a starting point for troubleshooting .NET client applications, see [Debugging, Tracing, and Profiling](#).

### Investigating network latency issues

Typically, high end-to-end latency caused by the network is due to transient conditions. You can investigate both transient and persistent network issues such as dropped packets by using tools such as Wireshark.

## Metrics show low SuccessE2ELatency and low SuccessServerLatency but the client is experiencing high latency

In this scenario, the most likely cause is a delay in the storage request reaching the storage service. You should investigate why requests from the client are not making it through to the blob service.

One possible reason for the client delaying sending requests is that there are a limited number of available connections or threads.

Also check whether the client is performing multiple retries, and investigate the reason if it is. To determine whether the client is performing multiple retries, you can:

- Examine logs. If multiple retries are happening, you will see multiple operations with the same client request IDs.
- Examine the client logs. Verbose logging will indicate that a retry has occurred.

- Debug your code, and check the properties of the **OperationContext** object associated with the request. If the operation has retried, the **RequestResults** property will include multiple unique requests. You can also check the start and end times for each request.

If there are no issues in the client, you should investigate potential network issues such as packet loss. You can use tools such as Wireshark to investigate network issues.

## Metrics show high SuccessServerLatency

In the case of high **SuccessServerLatency** for blob download requests, you should use the Storage logs to see if there are repeated requests for the same blob (or set of blobs). For blob upload requests, you should investigate what block size the client is using (for example, blocks less than 64 K in size can result in overheads unless the reads are also in less than 64 K chunks), and if multiple clients are uploading blocks to the same blob in parallel. You should also check the per-minute metrics for spikes in the number of requests that result in exceeding the per second scalability targets.

If you are seeing high **SuccessServerLatency** for blob download requests when there are repeated requests to the same blob or set of blobs, then you should consider caching these blobs using Azure Cache or the Azure Content Delivery Network (CDN). For upload requests, you can improve the throughput by using a larger block size. For queries to tables, it is also possible to implement client-side caching on clients that perform the same query operations and where the data doesn't change frequently.

High **SuccessServerLatency** values can also be a symptom of poorly designed tables or queries that result in scan operations or that follow the append/prepend anti-pattern.

### NOTE

You can find a comprehensive checklist performance checklist here: [Microsoft Azure Storage Performance and Scalability Checklist](#).

## You are experiencing unexpected delays in message delivery on a queue

If you are experiencing a delay between the time an application adds a message to a queue and the time it becomes available to read from the queue, then you should take the following steps to diagnose the issue:

- Verify the application is successfully adding the messages to the queue. Check that the application is not retrying the **AddMessage** method several times before succeeding.
- Verify there is no clock skew between the worker role that adds the message to the queue and the worker role that reads the message from the queue that makes it appear as if there is a delay in processing.
- Check if the worker role that reads the messages from the queue is failing. If a queue client calls the **GetMessage** method but fails to respond with an acknowledgment, the message will remain invisible on the queue until the **invisibilityTimeout** period expires. At this point, the message becomes available for processing again.
- Check if the queue length is growing over time. This can occur if you do not have sufficient workers available to process all of the messages that other workers are placing on the queue. Also check the metrics to see if delete requests are failing and the dequeue count on messages, which might indicate repeated failed attempts to delete the message.
- Examine the Storage logs for any queue operations that have higher than expected **E2ELatency** and **ServerLatency** values over a longer period of time than usual.

## See also

- [Troubleshoot client application errors](#)
- [Troubleshoot availability issues](#)
- [Monitor, diagnose, and troubleshoot your Azure Storage](#)

# Troubleshoot availability issues in Azure Storage accounts

8/22/2022 • 6 minutes to read • [Edit Online](#)

This article helps you investigate changes in the availability (such as number of failed requests). These changes in availability can often be identified by monitoring storage metrics in Azure Monitor. For general information about using metrics and logs in Azure Monitor, see

- [Monitoring Azure Blob Storage](#)
- [Monitoring Azure Files](#)
- [Monitoring Azure Queue Storage](#)
- [Monitoring Azure Table storage](#)

## Monitoring availability

You should monitor the availability of the storage services in your storage account by monitoring the value of the **Availability** metric. The **Availability** metric contains a percentage value and is calculated by taking the total billable requests value and dividing it by the number of applicable requests, including those requests that produced unexpected errors.

Any value less than 100% indicates that some storage requests are failing. You can see why they are failing by examining the **ResponseType** dimension for error types such as **ServerTimeoutError**. You should expect to see **Availability** fall temporarily below 100% for reasons such as transient server timeouts while the service moves partitions to better load-balance request; the retry logic in your client application should handle such intermittent conditions.

You can use features in Azure Monitor to alert you if **Availability** for a service falls below a threshold that you specify.

## Metrics show an increase in throttling errors

Throttling errors occur when you exceed the scalability targets of a storage service. The storage service throttles to ensure that no single client or tenant can use the service at the expense of others. For more information, see [Scalability and performance targets for standard storage accounts](#) for details on scalability targets for storage accounts and performance targets for partitions within storage accounts.

If the **ClientThrottlingError** or **ServerBusyError** value of the **ResponseType** dimension shows an increase in the percentage of requests that are failing with a throttling error, you need to investigate one of two scenarios:

- Transient increase in PercentThrottlingError
- Permanent increase in PercentThrottlingError error

An increase in throttling errors often occurs at the same time as an increase in the number of storage requests, or when you are initially load testing your application. This may also manifest itself in the client as "503 Server Busy" or "500 Operation Timeout" HTTP status messages from storage operations.

### Transient increase in throttling errors

If you are seeing spikes in throttling errors that coincide with periods of high activity for the application, you implement an exponential (not linear) back-off strategy for retries in your client. Back-off retries reduce the immediate load on the partition and help your application to smooth out spikes in traffic. For more information

about how to implement retry policies using the Storage Client Library, see the [RetryOptions.MaxRetries](#) property.

**NOTE**

You may also see spikes in throttling errors that do not coincide with periods of high activity for the application: the most likely cause here is the storage service moving partitions to improve load balancing.

### Permanent increase in throttling errors

If you are seeing a consistently high value for throttling errors following a permanent increase in your transaction volumes, or when you are performing your initial load tests on your application, then you need to evaluate how your application is using storage partitions and whether it is approaching the scalability targets for a storage account. For example, if you are seeing throttling errors on a queue (which counts as a single partition), then you should consider using additional queues to spread the transactions across multiple partitions. If you are seeing throttling errors on a table, you need to consider using a different partitioning scheme to spread your transactions across multiple partitions by using a wider range of partition key values. One common cause of this issue is the prepend/append anti-pattern where you select the date as the partition key and then all data on a particular day is written to one partition: under load, this can result in a write bottleneck. Either consider a different partitioning design or evaluate whether using blob storage might be a better solution. Also check whether throttling is occurring as a result of spikes in your traffic and investigate ways of smoothing your pattern of requests.

If you distribute your transactions across multiple partitions, you must still be aware of the scalability limits set for the storage account. For example, if you used ten queues each processing the maximum of 2,000 1KB messages per second, you will be at the overall limit of 20,000 messages per second for the storage account. If you need to process more than 20,000 entities per second, you should consider using multiple storage accounts. You should also bear in mind that the size of your requests and entities has an impact on when the storage service throttles your clients: if you have larger requests and entities, you may be throttled sooner.

Inefficient query design can also cause you to hit the scalability limits for table partitions. For example, a query with a filter that only selects one percent of the entities in a partition but that scans all the entities in a partition will need to access each entity. Every entity read will count towards the total number of transactions in that partition; therefore, you can easily reach the scalability targets.

**NOTE**

Your performance testing should reveal any inefficient query designs in your application.

## Metrics show an increase in timeout errors

Timeout errors occur when the `ResponseType` dimension is equal to `ServerErrorTimeoutError` or `ClientTimeout`.

Your metrics show an increase in timeout errors for one of your storage services. At the same time, the client receives a high volume of "500 Operation Timeout" HTTP status messages from storage operations.

**NOTE**

You may see timeout errors temporarily as the storage service load balances requests by moving a partition to a new server.

The server timeouts (`ServerErrorTimeoutError`) are caused by an error on the server. The client timeouts (`ClientTimeout`) happen because an operation on the server has exceeded the timeout specified by the client; for example, a client using the Storage Client Library can set a timeout for an operation.

Server timeouts indicate a problem with the storage service that requires further investigation. You can use metrics to see if you are hitting the scalability limits for the service and to identify any spikes in traffic that might be causing this problem. If the problem is intermittent, it may be due to load-balancing activity in the service. If the problem is persistent and is not caused by your application hitting the scalability limits of the service, you should raise a support issue. For client timeouts, you must decide if the timeout is set to an appropriate value in the client and either change the timeout value set in the client or investigate how you can improve the performance of the operations in the storage service, for example by optimizing your table queries or reducing the size of your messages.

## Metrics show an increase in network errors

Network errors occur when the **ResponseType** dimension is equal to **NetworkError**. These occur when a storage service detects a network error when the client makes a storage request.

The most common cause of this error is a client disconnecting before a timeout expires in the storage service. Investigate the code in your client to understand why and when the client disconnects from the storage service. You can also use third-party network analysis tools to investigate network connectivity issues from the client.

## See also

- [Troubleshoot client application errors](#)
- [Troubleshoot performance issues](#)
- [Monitor, diagnose, and troubleshoot your Azure Storage](#)

# Troubleshoot client application errors in Azure Storage accounts

8/22/2022 • 12 minutes to read • [Edit Online](#)

This article helps you investigate client application errors by using metrics, [client side logs](#), and resource logs in Azure Monitor.

## Diagnosing errors

Users of your application may notify you of errors reported by the client application. Azure Monitor also records counts of different response types (**ResponseType** dimensions) from your storage services such as **NetworkError**, **ClientTimeoutError**, or **AuthorizationError**. While Azure Monitor only records counts of different error types, you can obtain more detail about individual requests by examining server-side, client-side, and network logs. Typically, the HTTP status code returned by the storage service will give an indication of why the request failed.

### NOTE

Remember that you should expect to see some intermittent errors: for example, errors due to transient network conditions, or application errors.

The following resources are useful for understanding storage-related status and error codes:

- [Common REST API Error Codes](#)
- [Blob Service Error Codes](#)
- [Queue Service Error Codes](#)
- [Table Service Error Codes](#)
- [File Service Error Codes](#)

## The client is receiving HTTP 403 (Forbidden) messages

If your client application is throwing HTTP 403 (Forbidden) errors, a likely cause is that the client is using an expired Shared Access Signature (SAS) when it sends a storage request (although other possible causes include clock skew, invalid keys, and empty headers).

The Storage Client Library for .NET enables you to collect client-side log data that relates to storage operations performed by your application. For more information, see [Client-side Logging with the .NET Storage Client Library](#).

The following table shows a sample from the client-side log generated by the Storage Client Library that illustrates this issue occurring:

SOURCE	VERBOSITY	VERBOSITY	CLIENT REQUEST ID	OPERATION TEXT
Microsoft.Azure.Storage	Information	3	85d077ab -...	Starting operation with location Primary per location mode PrimaryOnly.
Microsoft.Azure.Storage	Information	3	85d077ab -...	Starting synchronous request to < <a href="https://developer.mozilla.org/docs/Web/API/XMLHttpRequest/Synchronous_and_Asynchronous_Usage">https://developer.mozilla.org/docs/Web/API/XMLHttpRequest/Synchronous_and_Asynchronous_Usage</a> >
Microsoft.Azure.Storage	Information	3	85d077ab -...	Waiting for response.
Microsoft.Azure.Storage	Warning	2	85d077ab -...	Exception thrown while waiting for response: The remote server returned an error: (403) Forbidden.
Microsoft.Azure.Storage	Information	3	85d077ab -...	Response received. Status code = 403, Request ID = 9d67c04a-64ed-4b0d-0515-3b14bbcd63d, Content-MD5 = , ETag = .
Microsoft.Azure.Storage	Warning	2	85d077ab -...	Exception thrown during the operation: The remote server returned an error: (403) Forbidden..

Source	Verbosity	Verbosity	Client Request ID	Operation Text
Microsoft.Azure.Storage	Information	3	85d077ab -...	Checking if the operation should be retried. Retry count = 0, HTTP status code = 403, Exception = The remote server returned an error: (403) Forbidden..
Microsoft.Azure.Storage	Information	3	85d077ab -...	The next location has been set to Primary, based on the location mode.
Microsoft.Azure.Storage	Error	1	85d077ab -...	Retry policy did not allow for a retry. Failing with The remote server returned an error: (403) Forbidden.

In this scenario, you should investigate why the SAS token is expiring before the client sends the token to the server:

- Typically, you should not set a start time when you create a SAS for a client to use immediately. If there are small clock differences between the host generating the SAS using the current time and the storage service, then it is possible for the storage service to receive a SAS that is not yet valid.
- Do not set a very short expiry time on a SAS. Again, small clock differences between the host generating the SAS and the storage service can lead to a SAS apparently expiring earlier than anticipated.
- Does the version parameter in the SAS key (for example `sv=2015-04-05`) match the version of the Storage Client Library you are using? We recommend that you always use the latest version of the storage client library.
- If you regenerate your storage access keys, any existing SAS tokens may be invalidated. This issue may arise if you generate SAS tokens with a long expiry time for client applications to cache.

If you are using the Storage Client Library to generate SAS tokens, then it is easy to build a valid token. However, if you are using the Storage REST API and constructing the SAS tokens by hand, see [Delegating Access with a Shared Access Signature](#).

## The client is receiving HTTP 404 (Not found) messages

If the client application receives an HTTP 404 (Not found) message from the server, this implies that the object the client was attempting to use (such as an entity, table, blob, container, or queue) does not exist in the storage service. There are a number of possible reasons for this, such as:

- The client or another process previously deleted the object
- A Shared Access Signature (SAS) authorization issue
- Client-side JavaScript code does not have permission to access the object
- Network failure

### The client or another process previously deleted the object

In scenarios where the client is attempting to read, update, or delete data in a storage service it is usually easy to identify in the storage resource logs a previous operation that deleted the object in question from the storage service. Often, the log data shows that another user or process deleted the object. In the Azure Monitor logs (server-side) show when a client deleted an object.

In the scenario where a client is attempting to insert an object, it may not be immediately obvious why this results in an HTTP 404 (Not found) response given that the client is creating a new object. However, if the client is creating a blob it must be able to find the blob container; if the client is creating a message it must be able to find a queue, and if the client is adding a row it must be able to find the table.

You can use the client-side log from the Storage Client Library to gain a more detailed understanding of when the client sends specific requests to the storage service.

The following client-side log generated by the Storage Client library illustrates the problem when the client cannot find the container for the blob it is creating. This log includes details of the following storage operations:

Request ID	Operation
07b26a5d...	<code>DeleteIfExists</code> method to delete the blob container. Note that this operation includes a <code>HEAD</code> request to check for the existence of the container.
e2d06d78...	<code>CreateIfNotExists</code> method to create the blob container. Note that this operation includes a <code>HEAD</code> request that checks for the existence of the container. The <code>HEAD</code> returns a 404 message but continues.

REQUEST ID	OPERATION
de8b1c3c-...	UploadFromStream method to create the blob. The PUT request fails with a 404 message

Log entries:

REQUEST ID	OPERATION TEXT
07b26a5d-...	Starting synchronous request to https://domemaildist.blob.core.windows.net/azuremmblobcontainer.
07b26a5d-...	StringToSign = HEAD.....x-ms-client-request-id:07b26a5d-....x-ms-date:Tue, 03 Jun 2014 10:33:11 GMT.x-ms-version:2014-02-14./domemaildist/azuremmblobcontainer.restype:container.
07b26a5d-...	Waiting for response.
07b26a5d-...	Response received. Status code = 200, Request ID = eeead849-..., Content-MD5 = , ETag = "0x8D14D2DC63D059B".
07b26a5d-...	Response headers were processed successfully, proceeding with the rest of the operation.
07b26a5d-...	Downloading response body.
07b26a5d-...	Operation completed successfully.
07b26a5d-...	Starting synchronous request to https://domemaildist.blob.core.windows.net/azuremmblobcontainer.
07b26a5d-...	StringToSign = DELETE.....x-ms-client-request-id:07b26a5d-....x-ms-date:Tue, 03 Jun 2014 10:33:12 GMT.x-ms-version:2014-02-14./domemaildist/azuremmblobcontainer.restype:container.
07b26a5d-...	Waiting for response.
07b26a5d-...	Response received. Status code = 202, Request ID = Gab2a4cf-..., Content-MD5 = , ETag = .
07b26a5d-...	Response headers were processed successfully, proceeding with the rest of the operation.
07b26a5d-...	Downloading response body.
07b26a5d-...	Operation completed successfully.
e2d06d78-...	Starting asynchronous request to https://domemaildist.blob.core.windows.net/azuremmblobcontainer.
e2d06d78-...	.
e2d06d78-...	StringToSign = HEAD.....x-ms-client-request-id:e2d06d78-....x-ms-date:Tue, 03 Jun 2014 10:33:12 GMT.x-ms-version:2014-02-14./domemaildist/azuremmblobcontainer.restype:container.
e2d06d78-...	Waiting for response.
de8b1c3c-...	Starting synchronous request to https://domemaildist.blob.core.windows.net/azuremmblobcontainer/blobCreated.txt.
de8b1c3c-...	StringToSign = PUT.....4.qCmf+TQLPhq/YYK50mP9ZQ==.....x-ms-blob-type:BlockBlob.x-ms-client-request-id:de8b1c3c-....x-ms-date:Tue, 03 Jun 2014 10:33:12 GMT.x-ms-version:2014-02-14./domemaildist/azuremmblobcontainer/blobCreated.txt.
de8b1c3c-...	Preparing to write request data.
e2d06d78-...	Exception thrown while waiting for response: The remote server returned an error: (404) Not Found..
e2d06d78-...	Response received. Status code = 404, Request ID = 353ae3bc-..., Content-MD5 = , ETag = .
e2d06d78-...	Response headers were processed successfully, proceeding with the rest of the operation.
e2d06d78-...	Downloading response body.

REQUEST ID	OPERATION TEXT
e2d06d78-...	Operation completed successfully.
e2d06d78-...	Starting asynchronous request to https://domemaildist.blob.core.windows.net/azuremmblobcontainer.
e2d06d78-...	StringToSign = PUT...0.....x-ms-client-request-id:e2d06d78-....x-ms-date:Tue, 03 Jun 2014 10:33:12 GMT.x-ms-version:2014-02-14./domemaildist/azuremmblobcontainer.restype:container.
e2d06d78-...	Waiting for response.
de8b1c3c-...	Writing request data.
de8b1c3c-...	Waiting for response.
e2d06d78-...	Exception thrown while waiting for response: The remote server returned an error: (409) Conflict..
e2d06d78-...	Response received. Status code = 409, Request ID = c27da20e-..., Content-MD5 = , ETag = .
e2d06d78-...	Downloading error response body.
de8b1c3c-...	Exception thrown while waiting for response: The remote server returned an error: (404) Not Found..
de8b1c3c-...	Response received. Status code = 404, Request ID = 0eaeb3e-..., Content-MD5 = , ETag = .
de8b1c3c-...	Exception thrown during the operation: The remote server returned an error: (404) Not Found..
de8b1c3c-...	Retry policy did not allow for a retry. Failing with The remote server returned an error: (404) Not Found..
e2d06d78-...	Retry policy did not allow for a retry. Failing with The remote server returned an error: (409) Conflict..

In this example, the log shows that the client is interleaving requests from the `CreateIfNotExists` method (request ID e2d06d78...) with the requests from the `UploadFromStream` method (de8b1c3c-...). This interleaving happens because the client application is invoking these methods asynchronously. Modify the asynchronous code in the client to ensure that it creates the container before attempting to upload any data to a blob in that container. Ideally, you should create all your containers in advance.

#### A Shared Access Signature (SAS) authorization issue

If the client application attempts to use a SAS key that does not include the necessary permissions for the operation, the storage service returns an HTTP 404 (Not found) message to the client. At the same time, in Azure Monitor metrics, you will also see an **AuthorizationError** for the **ResponseType** dimension.

Investigate why your client application is attempting to perform an operation for which it has not been granted permissions.

#### Client-side JavaScript code does not have permission to access the object

If you are using a JavaScript client and the storage service is returning HTTP 404 messages, you check for the following JavaScript errors in the browser:

```
SEC7120: Origin http://localhost:56309 not found in Access-Control-Allow-Origin header.
SCRIPT7002: XMLHttpRequest: Network Error 0x80070005, Access is denied.
```

#### NOTE

You can use the F12 Developer Tools in Internet Explorer to trace the messages exchanged between the browser and the storage service when you are troubleshooting client-side JavaScript issues.

These errors occur because the web browser implements the [same origin policy](#) security restriction that prevents a web page from calling an API in a different domain from the domain the page comes from.

To work around the JavaScript issue, you can configure Cross Origin Resource Sharing (CORS) for the storage service the client is accessing. For more information, see [Cross-Origin Resource Sharing \(CORS\) Support for Azure Storage Services](#).

The following code sample shows how to configure your blob service to allow JavaScript running in the Contoso domain to access a blob in your blob storage service:

- [.NET v12 SDK](#)

```

var connectionString = Constants.connectionString;

BlobServiceClient blobServiceClient = new BlobServiceClient(connectionString);

BlobServiceProperties sp = blobServiceClient.GetProperties();

// Set the service properties.
sp.DefaultServiceVersion = "2013-08-15";
BlobCorsRule bcr = new BlobCorsRule();
bcr.AllowedHeaders = "*";

bcr.AllowedMethods = "GET,POST";
bcr.AllowedOrigins = "http://www.contoso.com";
bcr.ExposedHeaders = "x-ms-*";
bcr.MaxAgeInSeconds = 5;
sp.Cors.Clear();
sp.Cors.Add(bcr);
blobServiceClient.SetProperties(sp);

```

### Network Failure

In some circumstances, lost network packets can lead to the storage service returning HTTP 404 messages to the client. For example, when your client application is deleting an entity from the table service you see the client throw a storage exception reporting an "HTTP 404 (Not Found)" status message from the table service. When you investigate the table in the table storage service, you see that the service did delete the entity as requested.

The exception details in the client include the request ID (7e84f12d...) assigned by the table service for the request; you can use this information to locate the request details in the storage resource logs in Azure Monitor by searching in [Fields that describe how the operation was authenticated](#) of log entries. You could also use the metrics to identify when failures such as this occur and then search the log files based on the time the metrics recorded this error. This log entry shows that the delete failed with an "HTTP (404) Client Other Error" status message. The same log entry also includes the request ID generated by the client in the [client-request-id](#) column (813ea74f...).

The server-side log also includes another entry with the same [client-request-id](#) value (813ea74f...) for a successful delete operation for the same entity, and from the same client. This successful delete operation took place very shortly before the failed delete request.

The most likely cause of this scenario is that the client sent a delete request for the entity to the table service, which succeeded, but did not receive an acknowledgment from the server (perhaps due to a temporary network issue). The client then automatically retried the operation (using the same [client-request-id](#)), and this retry failed because the entity had already been deleted.

If this problem occurs frequently, you should investigate why the client is failing to receive acknowledgments from the table service. If the problem is intermittent, you should trap the "HTTP (404) Not Found" error and log it in the client, but allow the client to continue.

### The client is receiving HTTP 409 (Conflict) messages

When a client deletes blob containers, tables, or queues there is a brief period before the name becomes available again. If the code in your client application deletes and then immediately recreates a blob container using the same name, the [CreateIfNotExists](#) method eventually fails with the HTTP 409 (Conflict) error.

The client application should use unique container names whenever it creates new containers if the delete/recreate pattern is common.

### Metrics show low PercentSuccess or analytics log entries have operations with transaction status of ClientOtherErrors

A [ResponseType](#) dimension equal to a value of [Success](#) captures the percent of operations that were successful based on their HTTP Status Code. Operations with status codes of 2XX count as successful, whereas operations with status codes in 3XX, 4XX and 5XX ranges are counted as unsuccessful and lower the Success metric value. In storage resource logs, these operations are recorded with a transaction status of [ClientOtherError](#).

It is important to note that these operations have completed successfully and therefore do not affect other metrics such as availability. Some examples of operations that execute successfully but that can result in unsuccessful HTTP status codes include:

- [ResourceNotFound](#) (Not Found 404), for example from a GET request to a blob that does not exist.
- [ResourceAlreadyExists](#) (Conflict 409), for example from a [CreateIfNotExist](#) operation where the resource already exists.
- [ConditionNotMet](#) (Not Modified 304), for example from a conditional operation such as when a client sends an [ETag](#) value and an [HTTP If-None-Match](#) header to request an image only if it has been updated since the last operation.

You can find a list of common REST API error codes that the storage services return on the page [Common REST API Error Codes](#).

### See also

- [Monitoring Azure Blob Storage](#)
- [Monitoring Azure Files](#)
- [Monitoring Azure Queue Storage](#)

- [Monitoring Azure Table storage](#)
- [Troubleshoot performance issues](#)
- [Troubleshoot availability issues](#)
- [Monitor, diagnose, and troubleshoot your Azure Storage](#)

# Enable and manage Azure Storage Analytics metrics (classic)

8/22/2022 • 10 minutes to read • [Edit Online](#)

Azure Storage Analytics provides metrics for all storage services for blobs, queues, and tables. You can use the [Azure portal](#) to configure which metrics are recorded for your account, and configure charts that provide visual representations of your metrics data. This article shows you how to enable and manage metrics. To learn how to enable logs, see [Enable and manage Azure Storage Analytics logs \(classic\)](#).

We recommend you review [Azure Monitor for Storage](#). It is a feature of Azure Monitor that offers comprehensive monitoring of your Azure Storage accounts by delivering a unified view of your Azure Storage services performance, capacity, and availability. It does not require you to enable or configure anything, and you can immediately view these metrics from the pre-defined interactive charts and other visualizations included.

## NOTE

There are costs associated with examining monitoring data in the Azure portal. For more information, see [Storage Analytics](#).

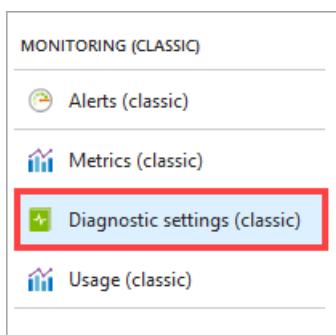
Premium performance block blob storage accounts don't support Storage Analytics metrics. If you want to view metrics with premium performance block blob storage accounts, consider using [Azure Storage Metrics in Azure Monitor](#).

For an in-depth guide on using Storage Analytics and other tools to identify, diagnose, and troubleshoot Azure Storage-related issues, see [Monitor, diagnose, and troubleshoot Microsoft Azure Storage](#).

## Enable metrics

- [Portal](#)
- [PowerShell](#)
- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

1. In the [Azure portal](#), select **Storage accounts**, then the storage account name to open the account dashboard.
2. Select **Diagnostic settings (classic)** in the **Monitoring (classic)** section of the menu blade.



3. Select the **type** of metrics data for each **service** you wish to monitor, and the **retention policy** for the data. You can also disable monitoring by setting **Status** to Off.

Status ⓘ

 [Blob properties](#)[File properties](#)[Table properties](#)[Queue properties](#)

## Hour metrics

 Enable Include API metrics Delete data after 7 days 7

## Minute metrics

 Enable Include API metrics Delete data after 7 days 7

## Logging

Logging version ⓘ

  Read Write Delete Delete data 7

To set the data retention policy, move the **Retention (days)** slider or enter the number of days of data to retain, from 1 to 365. The default for new storage accounts is seven days. If you do not want to set a retention policy, enter zero. If there is no retention policy, it is up to you to delete the monitoring data.

**WARNING**

Metrics are stored as data in your account. Metric data can accumulate in your account over time which can increase the cost of storage. If you need metric data for only a small period of time, you can reduce your costs by modifying the data retention policy. Stale metrics data (data older than your retention policy) is deleted by the system. We recommend setting a retention policy based on how long you want to retain the metrics data for your account. See [Billing on storage metrics](#) for more information.

- When you finish the monitoring configuration, select **Save**.

A default set of metrics is displayed in charts on the **Overview** blade, as well as the **Metrics (classic)** blade. Once you've enabled metrics for a service, it may take up to an hour for data to appear in its charts. You can select **Edit** on any metric chart to configure which metrics are displayed in the chart.

You can disable metrics collection and logging by setting **Status** to **Off**.

**NOTE**

Azure Storage uses [table storage](#) to store the metrics for your storage account, and stores the metrics in tables in your account. For more information, see [How metrics are stored](#).

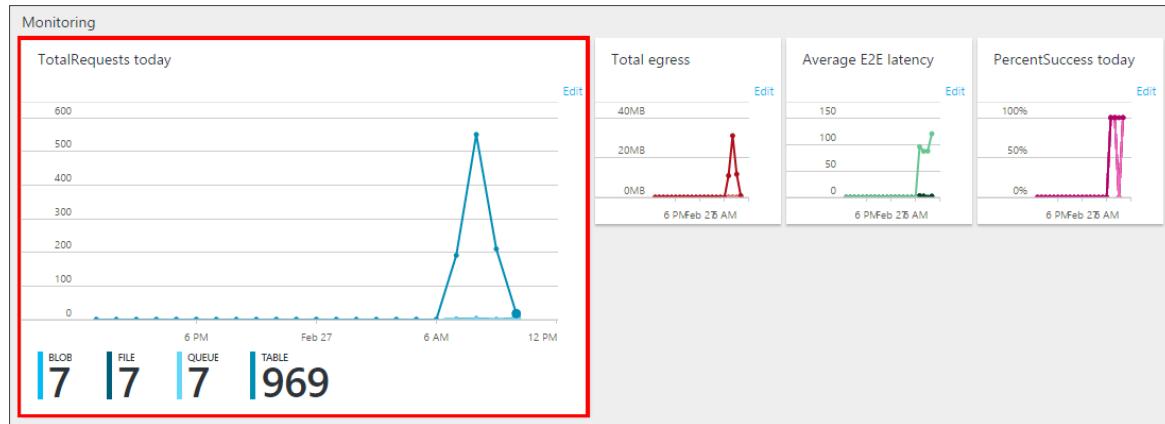
## View metrics in a chart

After you configure Storage Analytics metrics to monitor your storage account, Storage Analytics records the metrics in a set of well-known tables in your storage account. You can configure charts to view hourly metrics in the [Azure portal](#).

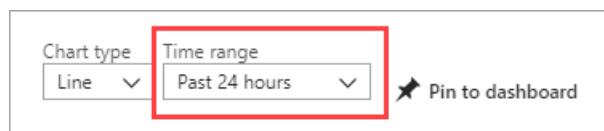
Use the following procedure to choose which storage metrics to view in a metrics chart.

1. Start by displaying a storage metric chart in the Azure portal. You can find charts on the **storage account blade** and in the **Metrics (classic)** blade.

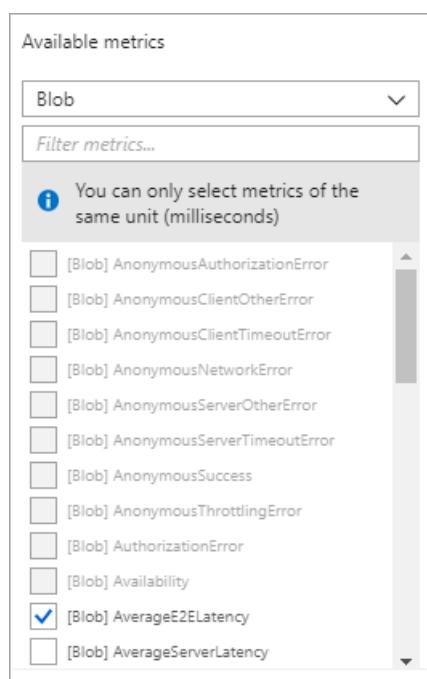
In this example, uses the following chart that appears on the **storage account blade**:



2. Click anywhere within the chart to edit the chart.
3. Next, select the **Time Range** of the metrics to display in the chart, and the **service** (blob, queue, table, file) whose metrics you wish to display. Here, the past week's metrics are selected to display for the blob service:



4. Select the individual **metrics** you'd like displayed in the chart, then click **OK**.

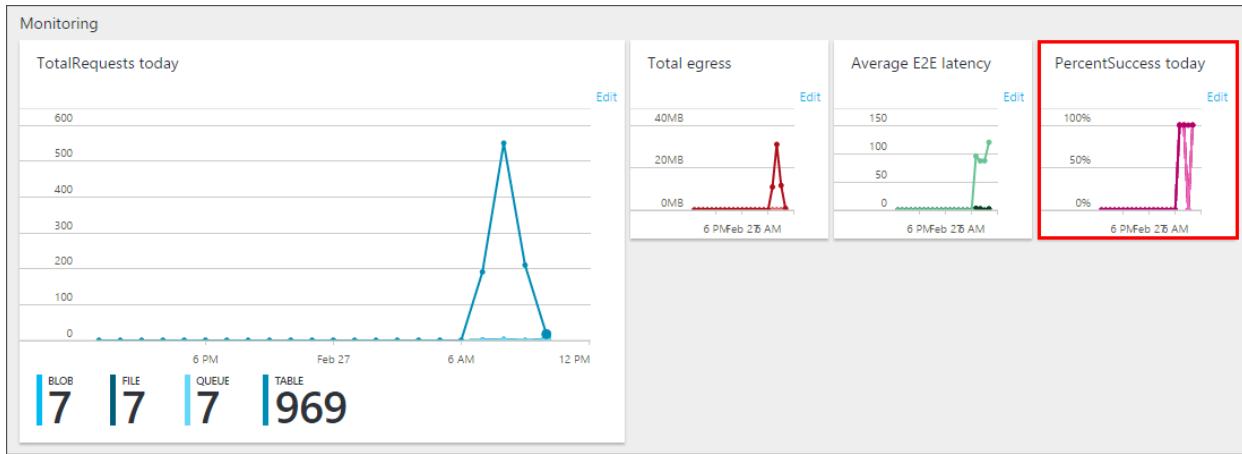


Your chart settings do not affect the collection, aggregation, or storage of monitoring data in the storage

account.

#### Metrics availability in charts

The list of available metrics changes based on which service you've chosen in the drop-down, and the unit type of the chart you're editing. For example, you can select percentage metrics like *PercentNetworkError* and *PercentThrottlingError* only if you're editing a chart that displays units in percentage:



#### Metrics resolution

The metrics you selected in **Diagnostics** determines the resolution of the metrics that are available for your account:

- **Aggregate** monitoring provides metrics such as ingress/egress, availability, latency, and success percentages. These metrics are aggregated from the blob, table, file, and queue services.
- **Per API** provides finer resolution, with metrics available for individual storage operations, in addition to the service-level aggregates.

## Download metrics to archive or analyze locally

If you want to download the metrics for long-term storage or to analyze them locally, you must use a tool or write some code to read the tables. The tables don't appear if you list all the tables in your storage account, but you can access them directly by name. Many storage-browsing tools are aware of these tables and enable you to view them directly. For a list of available tools, see [Azure Storage client tools](#).

METRICS	TABLE NAMES	NOTES
Hourly metrics	\$MetricsHourPrimaryTransactionsBlob \$MetricsHourPrimaryTransactionsTable \$MetricsHourPrimaryTransactionsQueue \$MetricsHourPrimaryTransactionsFile	In versions prior to August 15, 2013, these tables were known as:  \$MetricsTransactionsBlob \$MetricsTransactionsTable \$MetricsTransactionsQueue  Metrics for the file service are available beginning with version April 5, 2015.

METRICS	TABLE NAMES	NOTES
Minute metrics	\$MetricsMinutePrimaryTransactionsBlob \$MetricsMinutePrimaryTransactionsTable \$MetricsMinutePrimaryTransactionsQueue \$MetricsMinutePrimaryTransactionsFile	Can only be enabled by using PowerShell or programmatically.  Metrics for the file service are available beginning with version April 5, 2015.
Capacity	\$MetricsCapacityBlob	Blob service only.

For full details of the schemas for these tables, see [Storage Analytics metrics table schema](#). The following sample rows show only a subset of the columns available, but they illustrate some important features of the way storage metrics saves these metrics:

PARTITIONKEY	ROWKEY	TIMESTAMP	TOTALREQUESTS	TOTALBILLABLEREQUESTS	TOTALINGRESS	TOTALEGRESS	AVAILABILITY	AVERAGEEELATENCY	AVERAGERETRIVERTENCY	PERCENTSUCCESS
20140522T1100	user;All	2014-05-22T11:01:16.7650250Z	7	7	4003	46801	100	104.4286	6.857143	100
20140522T1100	user;QueryEntities	2014-05-22T11:01:16.7640250Z	5	5	2694	45951	100	143.8	7.8	100
20140522T1100	user;QueryEntity	2014-05-22T11:01:16.7650250Z	1	1	538	633	100	3	3	100
20140522T1100	user;UpdateEntity	2014-05-22T11:01:16.7650250Z	1	1	771	217	100	9	6	100

In this example of minute metrics data, the partition key uses the time at minute resolution. The row key identifies the type of information that's stored in the row. The information is composed of the access type and the request type:

- The access type is either **user** or **system**, where **user** refers to all user requests to the storage service and **system** refers to requests made by Storage Analytics.
- The request type is either **all**, in which case it's a summary line, or it identifies the specific API such as

## `QueryEntity` or `UpdateEntity`.

This sample data shows all the records for a single minute (starting at 11:00AM), so the number of `QueryEntities` requests plus the number of `QueryEntity` requests plus the number of `UpdateEntity` requests adds up to seven. This total is shown in the `user>All` row. Similarly, you can derive the average end-to-end latency 104.4286 on the `userAll` row by calculating  $((143.8 * 5) + 3 + 9)/7$ .

## View metrics data programmatically

The following listing shows sample C# code that accesses the minute metrics for a range of minutes and displays the results in a console window. The code sample uses the Azure Storage client library version 4.x or later, which includes the `CloudAnalyticsClient` class that simplifies accessing the metrics tables in storage.

### NOTE

The `CloudAnalyticsClient` class is not included in the Azure Blob storage client library v12 for .NET. On **August 31, 2023** Storage Analytics metrics, also referred to as *classic metrics* will be retired. For more information, see the [official announcement](#). If you use classic metrics, we recommend that you transition to metrics in Azure Monitor prior to that date.

```

private static void PrintMinuteMetrics(CloudAnalyticsClient analyticsClient, DateTimeOffset startTime,
DateTimeOffset endTime)
{
 // Convert the dates to the format used in the PartitionKey.
 var start = startTime.UniversalTime.ToString("yyyyMMdd'T'HHmm");
 var end = endTime.UniversalTime.ToString("yyyyMMdd'T'HHmm");

 var services = Enum.GetValues(typeof(StorageService));
 foreach (StorageService service in services)
 {
 Console.WriteLine("Minute Metrics for Service {0} from {1} to {2} UTC", service, start, end);
 var metricsQuery = analyticsClient.CreateMinuteMetricsQuery(service, StorageLocation.Primary);
 var t = analyticsClient.GetMinuteMetricsTable(service);
 var opContext = new OperationContext();
 var query =
 from entity in metricsQuery
 // Note, you can't filter using the entity properties Time, AccessType, or TransactionType
 // because they are calculated fields in the MetricsEntity class.
 // The PartitionKey identifies the DateTime of the metrics.
 where entity.PartitionKey.CompareTo(start) >= 0 && entity.PartitionKey.CompareTo(end) <= 0
 select entity;

 // Filter on "user" transactions after fetching the metrics from Azure Table storage.
 // (StartsWith is not supported using LINQ with Azure Table storage.)
 var results = query.ToList().Where(m => m.RowKey.StartsWith("user"));
 var resultString = results.Aggregate(new StringBuilder(), (builder, metrics) =>
builder.AppendLine(MetricsString(metrics, opContext))).ToString();
 Console.WriteLine(resultString);
 }
}

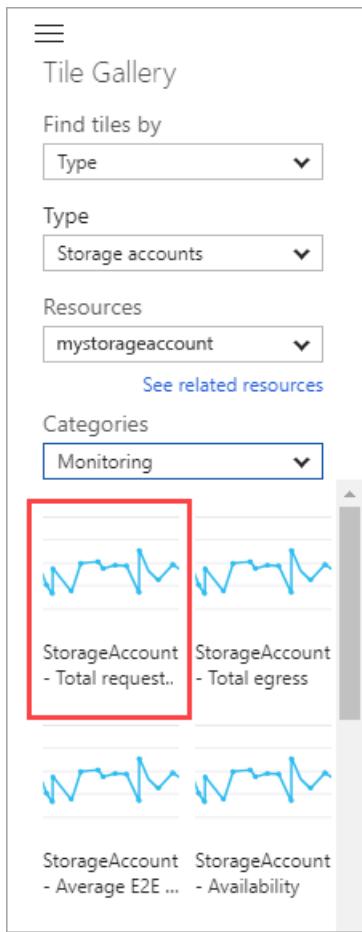
private static string MetricsString(MetricsEntity entity, OperationContext opContext)
{
 var entityProperties = entity.WriteEntity(opContext);
 var entityString =
 string.Format("Time: {0}, ", entity.Time) +
 string.Format("AccessType: {0}, ", entity.AccessType) +
 string.Format("TransactionType: {0}, ", entity.TransactionType) +
 string.Join(", ", entityProperties.Select(e => new KeyValuePair<string, string>(e.Key.ToString(),
e.Value.PropertyAsObject.ToString())));
 return entityString;
}

```

## Add metrics charts to the portal dashboard

You can add Azure Storage metrics charts for any of your storage accounts to your portal dashboard.

1. Select click **Edit dashboard** while viewing your dashboard in the [Azure portal](#).
2. In the **Tile Gallery**, select **Find tiles by > Type**.
3. Select **Type > Storage accounts**.
4. In **Resources**, select the storage account whose metrics you wish to add to the dashboard.
5. Select **Categories > Monitoring**.
6. Drag-and-drop the chart tile onto your dashboard for the metric you'd like displayed. Repeat for all metrics you'd like displayed on the dashboard. In the following image, the "Blobs - Total requests" chart is highlighted as an example, but all the charts are available for placement on your dashboard.



7. Select **Done customizing** near the top of the dashboard when you're done adding charts.

Once you've added charts to your dashboard, you can further customize them as described in [Customize metrics charts](#).

## Next steps

- To learn more about Storage Analytics, see [Storage Analytics](#) for Storage Analytics.
- [Configure Storage Analytics logs](#).
- Learn more about the the metrics schema. See [Storage Analytics metrics table schema](#).

# Enable and manage Azure Storage Analytics logs (classic)

8/22/2022 • 9 minutes to read • [Edit Online](#)

Azure Storage Analytics provides logs for blobs, queues, and tables. You can use the [Azure portal](#) to configure logs are recorded for your account. This article shows you how to enable and manage logs. To learn how to enable metrics, see [Enable and manage Azure Storage Analytics metrics \(classic\)](#). There are costs associated with examining and storing monitoring data in the Azure portal. For more information, see [Storage Analytics](#).

## NOTE

We recommend that you use Azure Storage logs in Azure Monitor instead of Storage Analytics logs. See any of the following articles:

- [Monitoring Azure Blob Storage](#)
- [Monitoring Azure Files](#)
- [Monitoring Azure Queue Storage](#)
- [Monitoring Azure Table storage](#)

## Enable logs

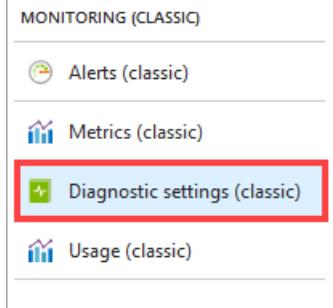
You can instruct Azure Storage to save diagnostics logs for read, write, and delete requests for the blob, table, and queue services. The data retention policy you set also applies to these logs.

## NOTE

Azure Files currently supports Storage Analytics metrics, but does not support Storage Analytics logging.

- [Portal](#)
- [PowerShell](#)
- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

1. In the [Azure portal](#), select **Storage accounts**, then the name of the storage account to open the storage account blade.
2. Select **Diagnostic settings (classic)** in the **Monitoring (classic)** section of the menu blade.



3. Ensure **Status** is set to **On**, and select the **services** for which you'd like to enable logging.



Status ⓘ

Off  On

Blob properties

File properties

Table properties

Queue properties

Hour metrics

Enable

Include API metrics

Delete data after 7 days

7

Minute metrics

Enable

Include API metrics

Delete data after 7 days

7

Logging

Logging version ⓘ

1.0

Read

Write

Delete

Delete data

7

4. Ensure that the **Delete data** check box is selected. Then, set the number of days that you would like log data to be retained by moving the slider control beneath the check box, or by directly modifying the value that appears in the text box next to the slider control. The default for new storage accounts is seven days. If you do not want to set a retention policy, enter zero. If there is no retention policy, it is up to you to delete the log data.

#### WARNING

Logs are stored as data in your account. Log data can accumulate in your account over time which can increase the cost of storage. If you need log data for only a small period of time, you can reduce your costs by modifying the data retention policy. Stale log data (data older than your retention policy) is deleted by the system. We recommend setting a retention policy based on how long you want to retain the log data for your account. See [Billing on storage metrics](#) for more information.

5. Click **Save**.

The diagnostics logs are saved in a blob container named `$logs` in your storage account. You can view the log data using a storage explorer like the [Microsoft Azure Storage Explorer](#), or programmatically using the storage client library or PowerShell.

For information about accessing the `$logs` container, see [Storage analytics logging](#).

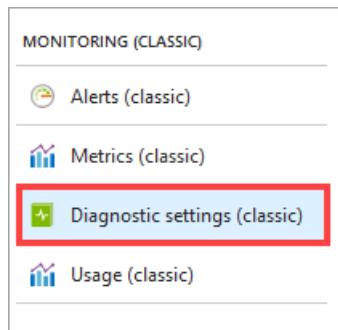
## Modify log data retention period

Log data can accumulate in your account over time which can increase the cost of storage. If you need log data for only a small period of time, you can reduce your costs by modifying the log data retention period. For example, if you need logs for only three days, set your log data retention period to a value of 3. That way logs will be automatically deleted from your account after 3 days. This section shows you how to view your current log data retention period, and then update that period if that's what you want to do.

- [Portal](#)
- [PowerShell](#)
- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

1. In the [Azure portal](#), select **Storage accounts**, then the name of the storage account to open the storage account blade.

2. Select **Diagnostic settings (classic)** in the **Monitoring (classic)** section of the menu blade.



3. Ensure that the **Delete data** check box is selected. Then, set the number of days that you would like log data to be retained by moving the slider control beneath the check box, or by directly modifying the value that appears in the text box next to the slider control.



Status ⓘ

Off  On

[Blob properties](#)

[File properties](#)

[Table properties](#)

[Queue properties](#)

Hour metrics

Enable

Include API metrics

Delete data after 7 days

7

Minute metrics

Enable

Include API metrics

Delete data after 7 days

7

Logging

Logging version ⓘ

1.0

Read

Write

Delete

Delete data

4

The default number of days for new storage accounts is seven days. If you do not want to set a retention policy, enter zero. If there is no retention policy, it is up to you to delete the monitoring data.

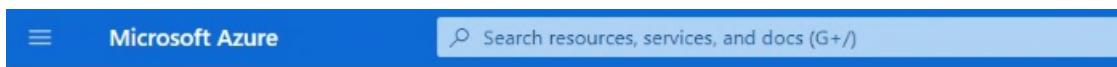
#### 4. Click Save.

The diagnostics logs are saved in a blob container named `$logs` in your storage account. You can view the log data using a storage explorer like the [Microsoft Azure Storage Explorer](#), or programmatically using the storage client library or PowerShell.

For information about accessing the `$logs` container, see [Storage analytics logging](#).

#### Verify that log data is being deleted

You can verify that logs are being deleted by viewing the contents of the `$logs` container of your storage account. The following image shows the contents of a folder in the `$logs` container. The folder corresponds to January 2021 and each folder contains logs for one day. If the day today was January 29th 2021, and your retention policy is set to only one day, then this folder should contain logs for only one day.



The Microsoft Azure Storage Explorer interface shows the '\$logs' container. The container name is highlighted with a red box. The interface includes a search bar, upload, change access level, refresh, and delete buttons. A sidebar on the left shows overview and access control options. The main area displays a list of blobs with names from '22' to '29'. A red box highlights this list.



\$logs

Container

Search (Ctrl+ /)

Upload

Change access level

Refresh

Delete

Overview

Access Control (IAM)

Settings

Access policy

Properties

Metadata

Authentication method: Azure AD User Account (Switch to Access key)

Location: \$logs / blob / 2021 / 01

Search blobs by prefix (case-sensitive)

#### Name

<input type="checkbox"/>	[..]
<input type="checkbox"/>	22
<input type="checkbox"/>	23
<input type="checkbox"/>	24
<input type="checkbox"/>	25
<input type="checkbox"/>	26
<input type="checkbox"/>	27
<input type="checkbox"/>	28
<input type="checkbox"/>	29

## View log data

To view and analyze your log data, you should download the blobs that contain the log data you are interested in to a local machine. Many storage-browsing tools enable you to download blobs from your storage account; you can also use the Azure Storage team provided command-line Azure Copy Tool [AzCopy](#) to download your log data.

#### NOTE

The '\$logs' container isn't integrated with Event Grid, so you won't receive notifications when log files are written.

To make sure you download the log data you are interested in and to avoid downloading the same log data more than once:

- Use the date and time naming convention for blobs containing log data to track which blobs you have already downloaded for analysis to avoid re-downloading the same data more than once.
- Use the metadata on the blobs containing log data to identify the specific period for which the blob holds log data to identify the exact blob you need to download.

To get started with AzCopy, see [Get started with AzCopy](#)

The following example shows how you can download the log data for the queue service for the hours starting at 09 AM, 10 AM, and 11 AM on 20th May, 2014.

```
azcopy copy 'https://mystorageaccount.blob.core.windows.net/$logs/queue' 'C:\Logs\Storage' --include-path
'2014/05/20/09;2014/05/20/10;2014/05/20/11' --recursive
```

To learn more about how to download specific files, see [Download blobs from Azure Blob storage by using AzCopy v10](#).

When you have downloaded your log data, you can view the log entries in the files. These log files use a delimited text format that many log reading tools are able to parse (for more information, see the guide [Monitoring, Diagnosing, and Troubleshooting Microsoft Azure Storage](#)). Different tools have different facilities for formatting, filtering, sorting, and searching the contents of your log files. For more information about the Storage Logging log file format and content, see [Storage Analytics Log Format](#) and [Storage Analytics Logged Operations and Status Messages](#).

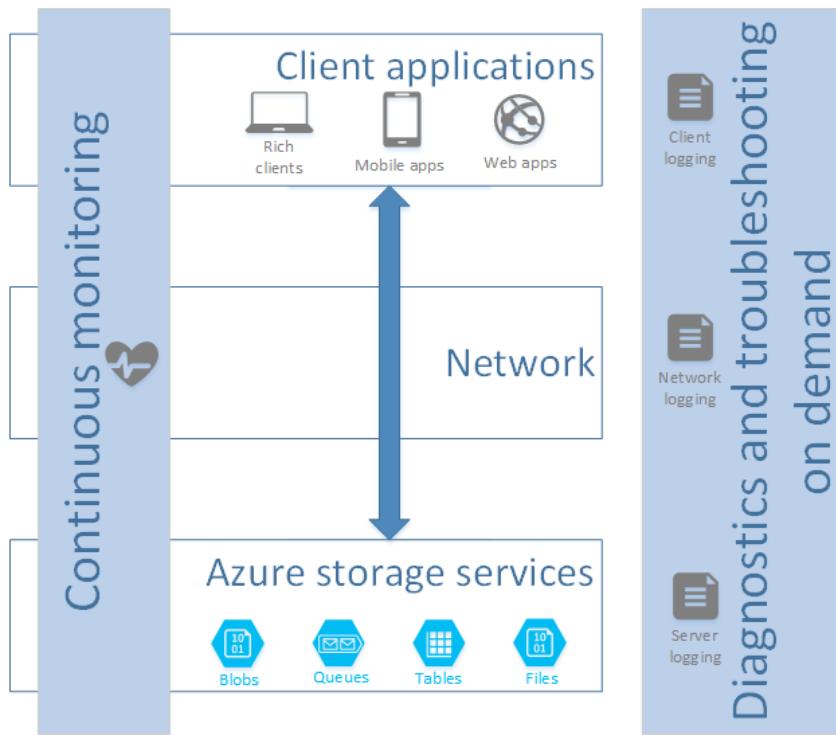
## Next steps

- To learn more about Storage Analytics, see [Storage Analytics](#) for Storage Analytics.
- For more information about using a .NET language to configure Storage Logging, see [Storage Client Library Reference](#).
- For general information about configuring Storage Logging using the REST API, see [Enabling and Configuring Storage Analytics](#).
- Learn more about the format of Storage Analytics logs. See [Storage Analytics Log Format](#).

# Monitor, diagnose, and troubleshoot Microsoft Azure Storage (classic)

8/22/2022 • 51 minutes to read • [Edit Online](#)

This guide shows you how to use features such as Azure Storage Analytics, client-side logging in the Azure Storage Client Library, and other third-party tools to identify, diagnose, and troubleshoot Azure Storage related issues.



This guide is intended to be read primarily by developers of online services that use Azure Storage Services and IT Pros responsible for managing such online services. The goals of this guide are:

- To help you maintain the health and performance of your Azure Storage accounts.
- To provide you with the necessary processes and tools to help you decide whether an issue or problem in an application relates to Azure Storage.
- To provide you with actionable guidance for resolving problems related to Azure Storage.

## NOTE

This article is based on using Storage Analytics metrics and logs as referred to as *Classic metrics and logs*. We recommend that you use Azure Storage metrics and logs in Azure Monitor instead of Storage Analytics logs. To learn more, see any of the following articles:

- [Monitoring Azure Blob Storage](#)
- [Monitoring Azure Files](#)
- [Monitoring Azure Queue Storage](#)
- [Monitoring Azure Table storage](#)

## Overview

Diagnosing and troubleshooting issues in a distributed application hosted in a cloud environment can be more complex than in traditional environments. Applications can be deployed in a PaaS or IaaS infrastructure, on-premises, on a mobile device, or in some combination of these environments. Typically, your application's network traffic may traverse public and private networks and your application may use multiple storage technologies such as Microsoft Azure Storage Tables, Blobs, Queues, or Files in addition to other data stores such as relational and document databases.

To manage such applications successfully you should monitor them proactively and understand how to diagnose and troubleshoot all aspects of them and their dependent technologies. As a user of Azure Storage services, you should continuously monitor the Storage services your application uses for any unexpected changes in behavior (such as slower than usual response times), and use logging to collect more detailed data and to analyze a problem in depth. The diagnostics information you obtain from both monitoring and logging will help you to determine the root cause of the issue your application encountered. Then you can troubleshoot the issue and determine the appropriate steps you can take to remediate it. Azure Storage is a core Azure service, and forms an important part of the majority of solutions that customers deploy to the Azure infrastructure. Azure Storage includes capabilities to simplify monitoring, diagnosing, and troubleshooting storage issues in your cloud-based applications.

### How this guide is organized

The section "[Monitoring your storage service](#)" describes how to monitor the health and performance of your Azure Storage services using Azure Storage Analytics Metrics (Storage Metrics).

The section "[Diagnosing storage issues](#)" describes how to diagnose issues using Azure Storage Analytics Logging (Storage Logging). It also describes how to enable client-side logging using the facilities in one of the client libraries such as the Storage Client Library for .NET or the Azure SDK for Java.

The section "[End-to-end tracing](#)" describes how you can correlate the information contained in various log files and metrics data.

The section "[Troubleshooting guidance](#)" provides troubleshooting guidance for some of the common storage-related issues you might encounter.

The "[Appendices](#)" include information about using other tools such as Wireshark and Netmon for analyzing network packet data, and Fiddler for analyzing HTTP/HTTPS messages.

## Monitoring your storage service

If you are familiar with Windows performance monitoring, you can think of Storage Metrics as being an Azure Storage equivalent of Windows Performance Monitor counters. In Storage Metrics, you will find a comprehensive set of metrics (counters in Windows Performance Monitor terminology) such as service availability, total number of requests to service, or percentage of successful requests to service. For a full list of the available metrics, see [Storage Analytics Metrics Table Schema](#). You can specify whether you want the storage service to collect and aggregate metrics every hour or every minute. For more information about how to enable metrics and monitor your storage accounts, see [Enabling storage metrics and viewing metrics data](#).

You can choose which hourly metrics you want to display in the [Azure portal](#) and configure rules that notify administrators by email whenever an hourly metric exceeds a particular threshold. For more information, see [Receive Alert Notifications](#).

We recommend you review [Azure Monitor for Storage](#) (preview). It is a feature of Azure Monitor that offers comprehensive monitoring of your Azure Storage accounts by delivering a unified view of your Azure Storage services performance, capacity, and availability. It does not require you to enable or configure anything, and you can immediately view these metrics from the pre-defined interactive charts and other visualizations included.

The storage service collects metrics using a best effort, but may not record every storage operation.

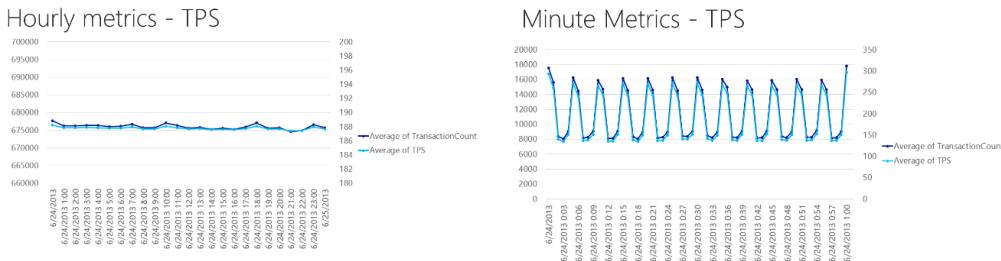
In the Azure portal, you can view metrics such as availability, total requests, and average latency numbers for a storage account. A notification rule has also been set up to alert an administrator if availability drops below a certain level. From viewing this data, one possible area for investigation is the table service success percentage being below 100% (for more information, see the section "[Metrics show low PercentSuccess or analytics log entries have operations with transaction status of ClientOtherErrors](#)").

You should continuously monitor your Azure applications to ensure they are healthy and performing as expected by:

- Establishing some baseline metrics for application that will enable you to compare current data and identify any significant changes in the behavior of Azure storage and your application. The values of your baseline metrics will, in many cases, be application specific and you should establish them when you are performance testing your application.
- Recording minute metrics and using them to monitor actively for unexpected errors and anomalies such as spikes in error counts or request rates.
- Recording hourly metrics and using them to monitor average values such as average error counts and request rates.

- Investigating potential issues using diagnostics tools as discussed later in the section "[Diagnosing storage issues](#)."

The charts in the following image illustrate how the averaging that occurs for hourly metrics can hide spikes in activity. The hourly metrics appear to show a steady rate of requests, while the minute metrics reveal the fluctuations that are really taking place.



The remainder of this section describes what metrics you should monitor and why.

### Monitoring service health

You can use the [Azure portal](#) to view the health of the Storage service (and other Azure services) in all the Azure regions around the world. Monitoring enables you to see immediately if an issue outside of your control is affecting the Storage service in the region you use for your application.

The [Azure portal](#) can also provide notifications of incidents that affect the various Azure services. Note: This information was previously available, along with historical data, on the [Azure Service Dashboard](#). For more information about Application Insights for Azure DevOps, see the appendix "[Appendix 5: Monitoring with Application Insights for Azure DevOps](#)."

### Monitoring capacity

Storage Metrics only stores capacity metrics for the blob service because blobs typically account for the largest proportion of stored data (at the time of writing, it is not possible to use Storage Metrics to monitor the capacity of your tables and queues). You can find this data in the **\$MetricsCapacityBlob** table if you have enabled monitoring for the Blob service. Storage Metrics records this data once per day, and you can use the value of the **RowKey** to determine whether the row contains an entity that relates to user data (value **data**) or analytics data (value **analytics**). Each stored entity contains information about the amount of storage used (**Capacity** measured in bytes) and the current number of containers (**ContainerCount**) and blobs (**ObjectCount**) in use in the storage account. For more information about the capacity metrics stored in the **\$MetricsCapacityBlob** table, see [Storage Analytics Metrics Table Schema](#).

#### NOTE

You should monitor these values for an early warning that you are approaching the capacity limits of your storage account. In the Azure portal, you can add alert rules to notify you if aggregate storage use exceeds or falls below thresholds that you specify.

For help estimating the size of various storage objects such as blobs, see the blog post [Understanding Azure Storage Billing – Bandwidth, Transactions, and Capacity](#).

### Monitoring availability

You should monitor the availability of the storage services in your storage account by monitoring the value in the **Availability** column in the hourly or minute metrics tables — **\$MetricsHourPrimaryTransactionsBlob**, **\$MetricsHourPrimaryTransactionsTable**, **\$MetricsHourPrimaryTransactionsQueue**, **\$MetricsMinutePrimaryTransactionsBlob**, **\$MetricsMinutePrimaryTransactionsTable**, **\$MetricsMinutePrimaryTransactionsQueue**, **\$MetricsCapacityBlob**. The **Availability** column contains a percentage value that indicates the availability of the service or the API operation represented by the row (the **RowKey** shows if the row contains metrics for the service as a whole or for a specific API operation).

Any value less than 100% indicates that some storage requests are failing. You can see why they are failing by examining the other columns in the metrics data that show the numbers of requests with different error types such as **ServerTimeoutError**. You should expect to see **Availability** fall temporarily below 100% for reasons such as transient server timeouts while the service moves partitions to better load-balance request; the retry logic in your client application should handle such intermittent conditions. The article [Storage Analytics Logged Operations and Status Messages](#) lists the transaction types that Storage Metrics includes in its **Availability** calculation.

In the [Azure portal](#), you can add alert rules to notify you if **Availability** for a service falls below a threshold that you specify.

The "Troubleshooting guidance" section of this guide describes some common storage service issues related to availability.

### Monitoring performance

To monitor the performance of the storage services, you can use the following metrics from the hourly and minute metrics tables.

- The values in the **AverageE2ELatency** and **AverageServerLatency** columns show the average time the storage service or API operation type is taking to process requests. **AverageE2ELatency** is a measure of end-to-end latency that includes the time taken to read the request and send the response in addition to the time taken to process the request (therefore includes network latency once the request reaches the storage service); **AverageServerLatency** is a measure of just the processing time and therefore excludes any network latency related to communicating with the client. See the section "[Metrics show high AverageE2ELatency and low AverageServerLatency](#)" later in this guide for a discussion of why there might be a significant difference between these two values.
- The values in the **TotalIngress** and **TotalEgress** columns show the total amount of data, in bytes, coming in to and going out of your storage service or through a specific API operation type.
- The values in the **TotalRequests** column show the total number of requests that the storage service or API operation is receiving. **TotalRequests** is the total number of requests that the storage service receives.

Typically, you will monitor for unexpected changes in any of these values as an indicator that you have an issue that requires investigation.

In the [Azure portal](#), you can add alert rules to notify you if any of the performance metrics for this service fall below or exceed a threshold that you specify.

The "Troubleshooting guidance" section of this guide describes some common storage service issues related to performance.

## Diagnosing storage issues

There are a number of ways that you might become aware of a problem or issue in your application, including:

- A major failure that causes the application to crash or to stop working.
- Significant changes from baseline values in the metrics you are monitoring as described in the previous section "[Monitoring your storage service](#)."
- Reports from users of your application that some particular operation didn't complete as expected or that some feature is not working.
- Errors generated within your application that appear in log files or through some other notification method.

Typically, issues related to Azure storage services fall into one of four broad categories:

- Your application has a performance issue, either reported by your users, or revealed by changes in the performance metrics.
- There is a problem with the Azure Storage infrastructure in one or more regions.
- Your application is encountering an error, either reported by your users, or revealed by an increase in one of the error count metrics you monitor.
- During development and test, you may be using the local storage emulator; you may encounter some issues that relate specifically to usage of the storage emulator.

The following sections outline the steps you should follow to diagnose and troubleshoot issues in each of these four categories. The section "[Troubleshooting guidance](#)" later in this guide provides more detail for some common issues you may encounter.

### Service health issues

Service health issues are typically outside of your control. The [Azure portal](#) provides information about any ongoing issues with Azure services including storage services. If you opted for Read-Access Geo-Redundant Storage when you created your storage account, then if your data becomes unavailable in the primary location, your application can switch temporarily to the read-only copy in the secondary location. To read from the secondary, your application must be able to switch between using the primary and secondary storage locations, and be able to work in a reduced functionality mode with read-only data. The Azure Storage Client libraries allow you to define a retry policy that can read from secondary storage in case a read from primary storage

fails. Your application also needs to be aware that the data in the secondary location is eventually consistent. For more information, see the blog post [Azure Storage Redundancy Options and Read Access Geo Redundant Storage](#).

### Performance issues

The performance of an application can be subjective, especially from a user perspective. Therefore, it is important to have baseline metrics available to help you identify where there might be a performance issue. Many factors might affect the performance of an Azure storage service from the client application perspective. These factors might operate in the storage service, in the client, or in the network infrastructure; therefore it is important to have a strategy for identifying the origin of the performance issue.

After you have identified the likely location of the cause of the performance issue from the metrics, you can then use the log files to find detailed information to diagnose and troubleshoot the problem further.

The section "[Troubleshooting guidance](#)" later in this guide provides more information about some common performance-related issues you may encounter.

### Diagnosing errors

Users of your application may notify you of errors reported by the client application. Storage Metrics also records counts of different error types from your storage services such as **NetworkError**, **ClientTimeoutError**, or **AuthorizationError**. While Storage Metrics only records counts of different error types, you can obtain more detail about individual requests by examining server-side, client-side, and network logs. Typically, the HTTP status code returned by the storage service will give an indication of why the request failed.

#### NOTE

Remember that you should expect to see some intermittent errors: for example, errors due to transient network conditions, or application errors.

The following resources are useful for understanding storage-related status and error codes:

- [Common REST API Error Codes](#)
- [Blob Service Error Codes](#)
- [Queue Service Error Codes](#)
- [Table Service Error Codes](#)
- [File Service Error Codes](#)

### Storage emulator issues

The Azure SDK includes a storage emulator you can run on a development workstation. This emulator simulates most of the behavior of the Azure storage services and is useful during development and test, enabling you to run applications that use Azure storage services without the need for an Azure subscription and an Azure storage account.

The "[Troubleshooting guidance](#)" section of this guide describes some common issues encountered using the storage emulator.

### Storage logging tools

Storage Logging provides server-side logging of storage requests in your Azure storage account. For more information about how to enable server-side logging and access the log data, see [Enabling Storage Logging and Accessing Log Data](#).

The Storage Client Library for .NET enables you to collect client-side log data that relates to storage operations performed by your application. For more information, see [Client-side Logging with the .NET Storage Client Library](#).

#### NOTE

In some circumstances (such as SAS authorization failures), a user may report an error for which you can find no request data in the server-side Storage logs. You can use the logging capabilities of the Storage Client Library to investigate if the cause of the issue is on the client or use network monitoring tools to investigate the network.

### Using network logging tools

You can capture the traffic between the client and server to provide detailed information about the data the client and server are exchanging and the underlying network conditions. Useful network logging tools include:

- [Fiddler](#) is a free web debugging proxy that enables you to examine the headers and payload data of HTTP and HTTPS request and response messages. For more information, see [Appendix 1: Using Fiddler to capture HTTP and HTTPS traffic](#).
- [Microsoft Network Monitor \(Netmon\)](#) and [Wireshark](#) are free network protocol analyzers that enable you to view detailed packet information for a wide range of network protocols. For more information about Wireshark, see ["Appendix 2: Using Wireshark to capture network traffic"](#).
- If you want to perform a basic connectivity test to check that your client machine can connect to the Azure storage service over the network, you cannot do this using the standard `ping` tool on the client. However, you can use the [tcping tool](#) to check connectivity.

In many cases, the log data from Storage Logging and the Storage Client Library will be sufficient to diagnose an issue, but in some scenarios, you may need the more detailed information that these network logging tools can provide. For example, using Fiddler to view HTTP and HTTPS messages enables you to view header and payload data sent to and from the storage services, which would enable you to examine how a client application retries storage operations. Protocol analyzers such as Wireshark operate at the packet level enabling you to view TCP data, which would enable you to troubleshoot lost packets and connectivity issues.

## End-to-end tracing

End-to-end tracing using a variety of log files is a useful technique for investigating potential issues. You can use the date/time information from your metrics data as an indication of where to start looking in the log files for the detailed information that will help you troubleshoot the issue.

### Correlating log data

When viewing logs from client applications, network traces, and server-side storage logging it is critical to be able to correlate requests across the different log files. The log files include a number of different fields that are useful as correlation identifiers. The client request ID is the most useful field to use to correlate entries in the different logs. However sometimes, it can be useful to use either the server request ID or timestamps. The following sections provide more details about these options.

### Client request ID

The Storage Client Library automatically generates a unique client request ID for every request.

- In the client-side log that the Storage Client Library creates, the client request ID appears in the **Client Request ID** field of every log entry relating to the request.
- In a network trace such as one captured by Fiddler, the client request ID is visible in request messages as the `x-ms-client-request-id` HTTP header value.
- In the server-side Storage Logging log, the client request ID appears in the Client request ID column.

#### NOTE

It is possible for multiple requests to share the same client request ID because the client can assign this value (although the Storage Client Library assigns a new value automatically). When the client retries, all attempts share the same client request ID. In the case of a batch sent from the client, the batch has a single client request ID.

### Server request ID

The storage service automatically generates server request IDs.

- In the server-side Storage Logging log, the server request ID appears the **Request ID header** column.
- In a network trace such as one captured by Fiddler, the server request ID appears in response messages as the `x-ms-request-id` HTTP header value.
- In the client-side log that the Storage Client Library creates, the server request ID appears in the **Operation Text** column for the log entry showing details of the server response.

#### NOTE

The storage service always assigns a unique server request ID to every request it receives, so every retry attempt from the client and every operation included in a batch has a unique server request ID.

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

The code sample below demonstrates how to use a custom client request ID.

```

var connectionString = Constants.connectionString;

BlobServiceClient blobServiceClient = new BlobServiceClient(connectionString);

BlobContainerClient blobContainerClient = blobServiceClient.GetBlobContainerClient("demcontainer");

BlobClient blobClient = blobContainerClient.GetBlobClient("testImage.jpg");

string clientRequestID = String.Format("{0} {1} {2} {3}", HOSTNAME, APPNAME, USERID,
Guid.NewGuid().ToString());

using (HttpPipeline.CreateClientRequestIdScope(clientRequestID))
{
 BlobDownloadInfo download = blobClient.Download();

 using (FileStream downloadFileStream = File.OpenWrite("C:\\\\testImage.jpg"))
 {
 download.Content.CopyTo(downloadFileStream);
 downloadFileStream.Close();
 }
}

```

## Timestamps

You can also use timestamps to locate related log entries, but be careful of any clock skew between the client and server that may exist. Search plus or minus 15 minutes for matching server-side entries based on the timestamp on the client. Remember that the blob metadata for the blobs containing metrics indicates the time range for the metrics stored in the blob. This time range is useful if you have many metrics blobs for the same minute or hour.

## Troubleshooting guidance

This section will help you with the diagnosis and troubleshooting of some of the common issues your application may encounter when using the Azure storage services. Use the list below to locate the information relevant to your specific issue.

### Troubleshooting Decision Tree

---

Does your issue relate to the performance of one of the storage services?

- Metrics show high AverageE2ELatency and low AverageServerLatency
  - Metrics show low AverageE2ELatency and low AverageServerLatency but the client is experiencing high latency
  - Metrics show high AverageServerLatency
  - You are experiencing unexpected delays in message delivery on a queue
- 

Does your issue relate to the availability of one of the storage services?

- Metrics show an increase in PercentThrottlingError
  - Metrics show an increase in PercentTimeoutError
  - Metrics show an increase in PercentNetworkError
- 

Is your client application receiving an HTTP 4XX (such as 404) response from a storage service?

- The client is receiving HTTP 403 (Forbidden) messages
  - The client is receiving HTTP 404 (Not found) messages
  - The client is receiving HTTP 409 (Conflict) messages
- 

Metrics show low PercentSuccess or analytics log entries have operations with transaction status of ClientOtherErrors

---

Capacity metrics show an unexpected increase in storage capacity usage

---

Your issue arises from using the storage emulator for development or test

---

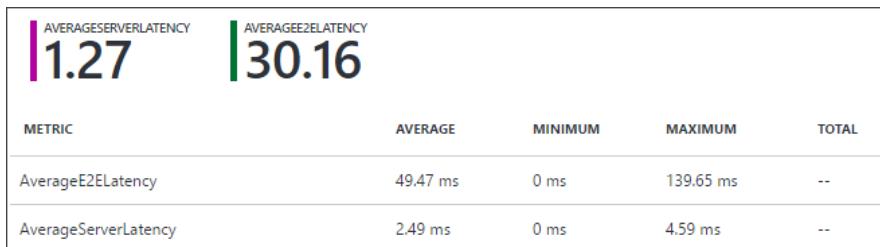
You are encountering problems installing the Azure SDK for .NET

---

You have a different issue with a storage service

### Metrics show high AverageE2ELatency and low AverageServerLatency

The illustration below from the [Azure portal](#) monitoring tool shows an example where the **AverageE2ELatency** is significantly higher than the **AverageServerLatency**.



The storage service only calculates the metric **AverageE2ELatency** for successful requests and, unlike **AverageServerLatency**, includes the time the client takes to send the data and receive acknowledgment from the storage service. Therefore, a difference between **AverageE2ELatency** and **AverageServerLatency** could be either due to the client application being slow to respond, or due to conditions on the network.

#### NOTE

You can also view **E2ELatency** and **ServerLatency** for individual storage operations in the Storage Logging log data.

#### Investigating client performance issues

Possible reasons for the client responding slowly include having a limited number of available connections or threads, or being low on resources such as CPU, memory or network bandwidth. You may be able to resolve the issue by modifying the client code to be more efficient (for example by using asynchronous calls to the storage service), or by using a larger Virtual Machine (with more cores and more memory).

For the table and queue services, the Nagle algorithm can also cause high **AverageE2ELatency** as compared to **AverageServerLatency**: for more information, see the post [Nagle's Algorithm is Not Friendly towards Small Requests](#). You can disable the Nagle algorithm in code by using the **ServicePointManager** class in the **System.Net** namespace. You should do this before you make any calls to the table or queue services in your application since this does not affect connections that are already open. The following example comes from the **Application\_Start** method in a worker role.

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

```
var connectionString = Constants.connectionString;
QueueServiceClient queueServiceClient = new QueueServiceClient(connectionString);
ServicePoint queueServicePoint = ServicePointManager.FindServicePoint(queueServiceClient.Uri);
queueServicePoint.UseNagleAlgorithm = false;
```

You should check the client-side logs to see how many requests your client application is submitting, and check for general .NET related performance bottlenecks in your client such as CPU, .NET garbage collection, network utilization, or memory. As a starting point for troubleshooting .NET client applications, see [Debugging, Tracing, and Profiling](#).

#### Investigating network latency issues

Typically, high end-to-end latency caused by the network is due to transient conditions. You can investigate both transient and persistent network issues such as dropped packets by using tools such as Wireshark.

For more information about using Wireshark to troubleshoot network issues, see "[Appendix 2: Using Wireshark to capture network traffic](#)."

### Metrics show low AverageE2ELatency and low AverageServerLatency but the client is experiencing high latency

In this scenario, the most likely cause is a delay in the storage requests reaching the storage service. You should investigate why requests from the client are not making it through to the blob service.

One possible reason for the client delaying sending requests is that there are a limited number of available

connections or threads.

Also check whether the client is performing multiple retries, and investigate the reason if it is. To determine whether the client is performing multiple retries, you can:

- Examine the Storage Analytics logs. If multiple retries are happening, you will see multiple operations with the same client request ID but with different server request IDs.
- Examine the client logs. Verbose logging will indicate that a retry has occurred.
- Debug your code, and check the properties of the `OperationContext` object associated with the request. If the operation has retried, the `RequestResults` property will include multiple unique server request IDs. You can also check the start and end times for each request. For more information, see the code sample in the section [Server request ID](#).

If there are no issues in the client, you should investigate potential network issues such as packet loss. You can use tools such as Wireshark to investigate network issues.

For more information about using Wireshark to troubleshoot network issues, see "[Appendix 2: Using Wireshark to capture network traffic](#)."

#### Metrics show high AverageServerLatency

In the case of high **AverageServerLatency** for blob download requests, you should use the Storage Logging logs to see if there are repeated requests for the same blob (or set of blobs). For blob upload requests, you should investigate what block size the client is using (for example, blocks less than 64 K in size can result in overheads unless the reads are also in less than 64 K chunks), and if multiple clients are uploading blocks to the same blob in parallel. You should also check the per-minute metrics for spikes in the number of requests that result in exceeding the per second scalability targets: also see "[Metrics show an increase in PercentTimeoutError](#)."

If you are seeing high **AverageServerLatency** for blob download requests when there are repeated requests to the same blob or set of blobs, then you should consider caching these blobs using Azure Cache or the Azure Content Delivery Network (CDN). For upload requests, you can improve the throughput by using a larger block size. For queries to tables, it is also possible to implement client-side caching on clients that perform the same query operations and where the data doesn't change frequently.

High **AverageServerLatency** values can also be a symptom of poorly designed tables or queries that result in scan operations or that follow the append/prepend anti-pattern. For more information, see "[Metrics show an increase in PercentThrottlingError](#)".

#### NOTE

You can find a comprehensive checklist performance checklist here: [Microsoft Azure Storage Performance and Scalability Checklist](#).

#### You are experiencing unexpected delays in message delivery on a queue

If you are experiencing a delay between the time an application adds a message to a queue and the time it becomes available to read from the queue, then you should take the following steps to diagnose the issue:

- Verify the application is successfully adding the messages to the queue. Check that the application is not retrying the `AddMessage` method several times before succeeding. The Storage Client Library logs will show any repeated retries of storage operations.
- Verify there is no clock skew between the worker role that adds the message to the queue and the worker role that reads the message from the queue that makes it appear as if there is a delay in processing.
- Check if the worker role that reads the messages from the queue is failing. If a queue client calls the `GetMessage` method but fails to respond with an acknowledgment, the message will remain invisible on the queue until the `visibilityTimeout` period expires. At this point, the message becomes available for processing again.
- Check if the queue length is growing over time. This can occur if you do not have sufficient workers available to process all of the messages that other workers are placing on the queue. Also check the metrics to see if delete requests are failing and the dequeue count on messages, which might indicate repeated failed attempts to delete the message.
- Examine the Storage Logging logs for any queue operations that have higher than expected **E2ELatency** and **ServerLatency** values over a longer period of time than usual.

#### Metrics show an increase in PercentThrottlingError

Throttling errors occur when you exceed the scalability targets of a storage service. The storage service throttles

to ensure that no single client or tenant can use the service at the expense of others. For more information, see [Scalability and performance targets for standard storage accounts](#) for details on scalability targets for storage accounts and performance targets for partitions within storage accounts.

If the **PercentThrottlingError** metric shows an increase in the percentage of requests that are failing with a throttling error, you need to investigate one of two scenarios:

- [Transient increase in PercentThrottlingError](#)
- [Permanent increase in PercentThrottlingError error](#)

An increase in **PercentThrottlingError** often occurs at the same time as an increase in the number of storage requests, or when you are initially load testing your application. This may also manifest itself in the client as "503 Server Busy" or "500 Operation Timeout" HTTP status messages from storage operations.

#### **Transient increase in PercentThrottlingError**

If you are seeing spikes in the value of **PercentThrottlingError** that coincide with periods of high activity for the application, you implement an exponential (not linear) back-off strategy for retries in your client. Back-off retries reduce the immediate load on the partition and help your application to smooth out spikes in traffic. For more information about how to implement retry policies using the Storage Client Library, see the [Microsoft.Azure.Storage.RetryPolicies namespace](#).

#### **NOTE**

You may also see spikes in the value of **PercentThrottlingError** that do not coincide with periods of high activity for the application: the most likely cause here is the storage service moving partitions to improve load balancing.

#### **Permanent increase in PercentThrottlingError error**

If you are seeing a consistently high value for **PercentThrottlingError** following a permanent increase in your transaction volumes, or when you are performing your initial load tests on your application, then you need to evaluate how your application is using storage partitions and whether it is approaching the scalability targets for a storage account. For example, if you are seeing throttling errors on a queue (which counts as a single partition), then you should consider using additional queues to spread the transactions across multiple partitions. If you are seeing throttling errors on a table, you need to consider using a different partitioning scheme to spread your transactions across multiple partitions by using a wider range of partition key values. One common cause of this issue is the prepend/append anti-pattern where you select the date as the partition key and then all data on a particular day is written to one partition: under load, this can result in a write bottleneck. Either consider a different partitioning design or evaluate whether using blob storage might be a better solution. Also check whether throttling is occurring as a result of spikes in your traffic and investigate ways of smoothing your pattern of requests.

If you distribute your transactions across multiple partitions, you must still be aware of the scalability limits set for the storage account. For example, if you used ten queues each processing the maximum of 2,000 1KB messages per second, you will be at the overall limit of 20,000 messages per second for the storage account. If you need to process more than 20,000 entities per second, you should consider using multiple storage accounts. You should also bear in mind that the size of your requests and entities has an impact on when the storage service throttles your clients: if you have larger requests and entities, you may be throttled sooner.

Inefficient query design can also cause you to hit the scalability limits for table partitions. For example, a query with a filter that only selects one percent of the entities in a partition but that scans all the entities in a partition will need to access each entity. Every entity read will count towards the total number of transactions in that partition; therefore, you can easily reach the scalability targets.

#### **NOTE**

Your performance testing should reveal any inefficient query designs in your application.

#### **Metrics show an increase in PercentTimeoutError**

Your metrics show an increase in **PercentTimeoutError** for one of your storage services. At the same time, the client receives a high volume of "500 Operation Timeout" HTTP status messages from storage operations.

#### **NOTE**

You may see timeout errors temporarily as the storage service load balances requests by moving a partition to a new server.

The **PercentTimeoutError** metric is an aggregation of the following metrics: **ClientTimeoutError**, **AnonymousClientTimeoutError**, **SASClientTimeoutError**, **ServerTimeoutError**, **AnonymousServerTimeoutError**, and **SASServerTimeoutError**.

The server timeouts are caused by an error on the server. The client timeouts happen because an operation on the server has exceeded the timeout specified by the client; for example, a client using the Storage Client Library can set a timeout for an operation by using the **ServerTimeout** property of the **QueueRequestOptions** class.

Server timeouts indicate a problem with the storage service that requires further investigation. You can use metrics to see if you are hitting the scalability limits for the service and to identify any spikes in traffic that might be causing this problem. If the problem is intermittent, it may be due to load-balancing activity in the service. If the problem is persistent and is not caused by your application hitting the scalability limits of the service, you should raise a support issue. For client timeouts, you must decide if the timeout is set to an appropriate value in the client and either change the timeout value set in the client or investigate how you can improve the performance of the operations in the storage service, for example by optimizing your table queries or reducing the size of your messages.

#### Metrics show an increase in PercentNetworkError

Your metrics show an increase in **PercentNetworkError** for one of your storage services. The **PercentNetworkError** metric is an aggregation of the following metrics: **NetworkError**, **AnonymousNetworkError**, and **SASNetworkError**. These occur when the storage service detects a network error when the client makes a storage request.

The most common cause of this error is a client disconnecting before a timeout expires in the storage service. Investigate the code in your client to understand why and when the client disconnects from the storage service. You can also use Wireshark, or Tcping to investigate network connectivity issues from the client. These tools are described in the [Appendices](#).

#### The client is receiving HTTP 403 (Forbidden) messages

If your client application is throwing HTTP 403 (Forbidden) errors, a likely cause is that the client is using an expired Shared Access Signature (SAS) when it sends a storage request (although other possible causes include clock skew, invalid keys, and empty headers). If an expired SAS key is the cause, you will not see any entries in the server-side Storage Logging log data. The following table shows a sample from the client-side log generated by the Storage Client Library that illustrates this issue occurring:

SOURCE	VERBOSITY	VERBOSITY	CLIENT REQUEST ID	OPERATION TEXT
Microsoft.Azure.Storage	Information	3	85d077ab-...	Starting operation with location Primary per location mode PrimaryOnly.
Microsoft.Azure.Storage	Information	3	85d077ab -...	Starting synchronous request to <a href="https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Synchronous_and_Asynchronous_Requests#Synchronous_request">https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Synchronous_and_Asynchronous_Requests#Synchronous_request</a>
Microsoft.Azure.Storage	Information	3	85d077ab -...	Waiting for response.
Microsoft.Azure.Storage	Warning	2	85d077ab -...	Exception thrown while waiting for response: The remote server returned an error: (403) Forbidden.

Source	Verbosity	Verbosity	Client Request ID	Operation Text
Microsoft.Azure.Storage	Information	3	85d077ab -...	Response received. Status code = 403, Request ID = 9d67c64a-64ed- 4b0d-9515- 3b14bbcd63d, Content-MD5 = , ETag = .
Microsoft.Azure.Storage	Warning	2	85d077ab -...	Exception thrown during the operation: The remote server returned an error: (403) Forbidden..
Microsoft.Azure.Storage	Information	3	85d077ab -...	Checking if the operation should be retried. Retry count = 0, HTTP status code = 403, Exception = The remote server returned an error: (403) Forbidden..
Microsoft.Azure.Storage	Information	3	85d077ab -...	The next location has been set to Primary, based on the location mode.
Microsoft.Azure.Storage	Error	1	85d077ab -...	Retry policy did not allow for a retry. Failing with The remote server returned an error: (403) Forbidden.

In this scenario, you should investigate why the SAS token is expiring before the client sends the token to the server:

- Typically, you should not set a start time when you create a SAS for a client to use immediately. If there are small clock differences between the host generating the SAS using the current time and the storage service, then it is possible for the storage service to receive a SAS that is not yet valid.
- Do not set a very short expiry time on a SAS. Again, small clock differences between the host generating the SAS and the storage service can lead to a SAS apparently expiring earlier than anticipated.
- Does the version parameter in the SAS key (for example `sv=2015-04-05`) match the version of the Storage Client Library you are using? We recommend that you always use the latest version of the [Storage Client Library](#).
- If you regenerate your storage access keys, any existing SAS tokens may be invalidated. This issue may arise if you generate SAS tokens with a long expiry time for client applications to cache.

If you are using the Storage Client Library to generate SAS tokens, then it is easy to build a valid token. However, if you are using the Storage REST API and constructing the SAS tokens by hand, see [Delegating Access with a Shared Access Signature](#).

#### The client is receiving HTTP 404 (Not found) messages

If the client application receives an HTTP 404 (Not found) message from the server, this implies that the object the client was attempting to use (such as an entity, table, blob, container, or queue) does not exist in the storage service. There are a number of possible reasons for this, such as:

- [The client or another process previously deleted the object](#)
- [A Shared Access Signature \(SAS\) authorization issue](#)
- [Client-side JavaScript code does not have permission to access the object](#)
- [Network failure](#)

#### The client or another process previously deleted the object

In scenarios where the client is attempting to read, update, or delete data in a storage service it is usually easy to

identify in the server-side logs a previous operation that deleted the object in question from the storage service. Often, the log data shows that another user or process deleted the object. In the server-side Storage Logging log, the operation-type and requested-object-key columns show when a client deleted an object.

In the scenario where a client is attempting to insert an object, it may not be immediately obvious why this results in an HTTP 404 (Not found) response given that the client is creating a new object. However, if the client is creating a blob it must be able to find the blob container; if the client is creating a message it must be able to find a queue, and if the client is adding a row it must be able to find the table.

You can use the client-side log from the Storage Client Library to gain a more detailed understanding of when the client sends specific requests to the storage service.

The following client-side log generated by the Storage Client library illustrates the problem when the client cannot find the container for the blob it is creating. This log includes details of the following storage operations:

REQUEST ID	OPERATION
07b26a5d...	<b>DeleteIfExists</b> method to delete the blob container. Note that this operation includes a <b>HEAD</b> request to check for the existence of the container.
e2d06d78...	<b>CreateIfNotExists</b> method to create the blob container. Note that this operation includes a <b>HEAD</b> request that checks for the existence of the container. The <b>HEAD</b> returns a 404 message but continues.
de8b1c3c...	<b>UploadFromStream</b> method to create the blob. The <b>PUT</b> request fails with a 404 message

Log entries:

REQUEST ID	OPERATION TEXT
07b26a5d...	Starting synchronous request to <code>https://domemaildist.blob.core.windows.net/azuremmblobcontainer</code>
07b26a5d...	StringToSign = HEAD.....x-ms-client-request-id:07b26a5d-...x-ms-date:Tue, 03 Jun 2014 10:33:11 GMTx-ms-version:2014-02-14./domemaildist/azuremmblobcontainer.restype:container.
07b26a5d...	Waiting for response.
07b26a5d...	Response received. Status code = 200, Request ID = eeead849-...Content-MD5 = , ETag = "0x8D14D2DC63D059B".
07b26a5d...	Response headers were processed successfully, proceeding with the rest of the operation.
07b26a5d...	Downloading response body.
07b26a5d...	Operation completed successfully.
07b26a5d...	Starting synchronous request to <code>https://domemaildist.blob.core.windows.net/azuremmblobcontainer</code>
07b26a5d...	StringToSign = DELETE.....x-ms-client-request-id:07b26a5d-...x-ms-date:Tue, 03 Jun 2014 10:33:12 GMTx-ms-version:2014-02-14./domemaildist/azuremmblobcontainer.restype:container.
07b26a5d...	Waiting for response.
07b26a5d...	Response received. Status code = 202, Request ID = 6ab2a4cf..., Content-MD5 = , ETag = .

REQUEST ID	OPERATION TEXT
07b26a5d-...	Response headers were processed successfully, proceeding with the rest of the operation.
07b26a5d-...	Downloading response body.
07b26a5d-...	Operation completed successfully.
e2d06d78-...	Starting asynchronous request to <code>https://domemaildist.blob.core.windows.net/azuremmblobcontainer</code>
e2d06d78-...	StringToSign = HEAD.....x-ms-client-request-id:e2d06d78-....x-ms-date:Tue, 03 Jun 2014 10:33:12 GMTx-ms-version:2014-02-14./domemaildist/azuremmblobcontainer.restype:container.
e2d06d78-...	Waiting for response.
de8b1c3c-...	Starting synchronous request to <code>https://domemaildist.blob.core.windows.net/azuremmblobcontainer/blobCreate</code>
de8b1c3c-...	StringToSign = PUT..64.qCmF+TQLPhq/YYK50mP9ZQ==.....x-ms-blob-type:BlockBlob.x-ms-client-request-id:de8b1c3c-....x-ms-date:Tue, 03 Jun 2014 10:33:12 GMTx-ms-version:2014-02-14./domemaildist/azuremmblobcontainer/blobCreated.txt.
de8b1c3c-...	Preparing to write request data.
e2d06d78-...	Exception thrown while waiting for response: The remote server returned an error: (404) Not Found..
e2d06d78-...	Response received. Status code = 404, Request ID = 353ae3bc-..., Content-MD5 = , ETag = .
e2d06d78-...	Response headers were processed successfully, proceeding with the rest of the operation.
e2d06d78-...	Downloading response body.
e2d06d78-...	Operation completed successfully.
e2d06d78-...	Starting asynchronous request to <code>https://domemaildist.blob.core.windows.net/azuremmblobcontainer</code>
e2d06d78-...	StringToSign = PUT..0.....x-ms-client-request-id:e2d06d78-....x-ms-date:Tue, 03 Jun 2014 10:33:12 GMTx-ms-version:2014-02-14./domemaildist/azuremmblobcontainer.restype:container.
e2d06d78-...	Waiting for response.
de8b1c3c-...	Writing request data.
de8b1c3c-...	Waiting for response.
e2d06d78-...	Exception thrown while waiting for response: The remote server returned an error: (409) Conflict..
e2d06d78-...	Response received. Status code = 409, Request ID = c27da20e-..., Content-MD5 = , ETag = .

REQUEST ID	OPERATION TEXT
e2d06d78-...	Downloading error response body.
de8b1c3c-...	Exception thrown while waiting for response: The remote server returned an error: (404) Not Found..
de8b1c3c-...	Response received. Status code = 404, Request ID = 0eaeb3e-..., Content-MD5 = , ETag = .
de8b1c3c-...	Exception thrown during the operation: The remote server returned an error: (404) Not Found..
de8b1c3c-...	Retry policy did not allow for a retry. Failing with The remote server returned an error: (404) Not Found..
e2d06d78-...	Retry policy did not allow for a retry. Failing with The remote server returned an error: (409) Conflict..

In this example, the log shows that the client is interleaving requests from the **CreateIfNotExist** method (request ID e2d06d78...) with the requests from the **UploadFromStream** method (de8b1c3c-...). This interleaving happens because the client application is invoking these methods asynchronously. Modify the asynchronous code in the client to ensure that it creates the container before attempting to upload any data to a blob in that container. Ideally, you should create all your containers in advance.

#### A Shared Access Signature (SAS) authorization issue

If the client application attempts to use a SAS key that does not include the necessary permissions for the operation, the storage service returns an HTTP 404 (Not found) message to the client. At the same time, you will also see a non-zero value for **SASAuthorizationError** in the metrics.

The following table shows a sample server-side log message from the Storage Logging log file:

NAME	VALUE
Request start time	2014-05-30T06:17:48.4473697Z
Operation type	GetBlobProperties
Request status	SASAuthorizationError
HTTP status code	404
Authentication type	Sas
Service type	Blob
Request URL	<a href="https://domemaildist.blob.core.windows.net/azureimblobcontainer/blobCreate">https://domemaildist.blob.core.windows.net/azureimblobcontainer/blobCreate</a>
	?sv=2014-02-14&sr=c&si=mypolicy&sig=XXXXX&api-version=2014-02-14
Request ID header	a1f348d5-8032-4912-93ef-b393e5252a3b
Client request ID	2d064953-8436-4ee0-aa0c-65cb874f7929

Investigate why your client application is attempting to perform an operation for which it has not been granted permissions.

#### Client-side JavaScript code does not have permission to access the object

If you are using a JavaScript client and the storage service is returning HTTP 404 messages, you check for the following JavaScript errors in the browser:

```
SEC7120: Origin http://localhost:56309 not found in Access-Control-Allow-Origin header.
SCRIPT7002: XMLHttpRequest: Network Error 0x80070005, Access is denied.
```

#### NOTE

You can use the F12 Developer Tools in Internet Explorer to trace the messages exchanged between the browser and the storage service when you are troubleshooting client-side JavaScript issues.

These errors occur because the web browser implements the [same origin policy](#) security restriction that prevents a web page from calling an API in a different domain from the domain the page comes from.

To work around the JavaScript issue, you can configure Cross Origin Resource Sharing (CORS) for the storage service the client is accessing. For more information, see [Cross-Origin Resource Sharing \(CORS\) Support for Azure Storage Services](#).

The following code sample shows how to configure your blob service to allow JavaScript running in the Contoso domain to access a blob in your blob storage service:

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

```
var connectionString = Constants.connectionString;

BlobServiceClient blobServiceClient = new BlobServiceClient(connectionString);

BlobServiceProperties sp = blobServiceClient.GetProperties();

// Set the service properties.
sp.DefaultServiceVersion = "2013-08-15";
BlobCorsRule bcr = new BlobCorsRule();
bcr.AllowedHeaders = "*";

bcr.AllowedMethods = "GET,POST";
bcr.AllowedOrigins = "http://www.contoso.com";
bcr.ExposedHeaders = "x-ms-*";
bcr.MaxAgeInSeconds = 5;
sp.Cors.Clear();
sp.Cors.Add(bcr);
blobServiceClient.SetProperties(sp);
```

#### Network Failure

In some circumstances, lost network packets can lead to the storage service returning HTTP 404 messages to the client. For example, when your client application is deleting an entity from the table service you see the client throw a storage exception reporting an "HTTP 404 (Not Found)" status message from the table service. When you investigate the table in the table storage service, you see that the service did delete the entity as requested.

The exception details in the client include the request ID (7e84f12d...) assigned by the table service for the request: you can use this information to locate the request details in the server-side storage logs by searching in the **request-id-header** column in the log file. You could also use the metrics to identify when failures such as this occur and then search the log files based on the time the metrics recorded this error. This log entry shows that the delete failed with an "HTTP (404) Client Other Error" status message. The same log entry also includes the request ID generated by the client in the **client-request-id** column (813ea74f...).

The server-side log also includes another entry with the same **client-request-id** value (813ea74f...) for a successful delete operation for the same entity, and from the same client. This successful delete operation took place very shortly before the failed delete request.

The most likely cause of this scenario is that the client sent a delete request for the entity to the table service, which succeeded, but did not receive an acknowledgment from the server (perhaps due to a temporary network issue). The client then automatically retried the operation (using the same **client-request-id**), and this retry failed because the entity had already been deleted.

If this problem occurs frequently, you should investigate why the client is failing to receive acknowledgments from the table service. If the problem is intermittent, you should trap the "HTTP (404) Not Found" error and log it in the client, but allow the client to continue.

#### The client is receiving HTTP 409 (Conflict) messages

The following table shows an extract from the server-side log for two client operations: **DeleteIfExists** followed immediately by **CreateIfNotExist** using the same blob container name. Each client operation results in two requests sent to the server, first a **GetContainerProperties** request to check if the container exists, followed by

the `DeleteContainer` or `CreateContainer` request.

Timestamp	Operation	Result	Container Name	Client Request ID
05:10:13.7167225	GetContainerProperties	200	mmcont	c9f52c89-...
05:10:13.8167325	DeleteContainer	202	mmcont	c9f52c89-...
05:10:13.8987407	GetContainerProperties	404	mmcont	bc881924-...
05:10:14.2147723	CreateContainer	409	mmcont	bc881924-...

The code in the client application deletes and then immediately recreates a blob container using the same name: the `CreateIfNotExists` method (Client request ID bc881924...) eventually fails with the HTTP 409 (Conflict) error. When a client deletes blob containers, tables, or queues there is a brief period before the name becomes available again.

The client application should use unique container names whenever it creates new containers if the delete/recreate pattern is common.

#### Metrics show low PercentSuccess or analytics log entries have operations with transaction status of ClientOtherErrors

The `PercentSuccess` metric captures the percent of operations that were successful based on their HTTP Status Code. Operations with status codes of 2XX count as successful, whereas operations with status codes in 3XX, 4XX and 5XX ranges are counted as unsuccessful and lower the `PercentSuccess` metric value. In the server-side storage log files, these operations are recorded with a transaction status of `ClientOtherErrors`.

It is important to note that these operations have completed successfully and therefore do not affect other metrics such as availability. Some examples of operations that execute successfully but that can result in unsuccessful HTTP status codes include:

- **ResourceNotFound** (Not Found 404), for example from a GET request to a blob that does not exist.
- **ResourceAlreadyExists** (Conflict 409), for example from a `CreateIfNotExist` operation where the resource already exists.
- **ConditionNotMet** (Not Modified 304), for example from a conditional operation such as when a client sends an `ETag` value and an HTTP `If-None-Match` header to request an image only if it has been updated since the last operation.

You can find a list of common REST API error codes that the storage services return on the page [Common REST API Error Codes](#).

#### Capacity metrics show an unexpected increase in storage capacity usage

If you see sudden, unexpected changes in capacity usage in your storage account, you can investigate the reasons by first looking at your availability metrics; for example, an increase in the number of failed delete requests might lead to an increase in the amount of blob storage you are using as application-specific cleanup operations you might have expected to be freeing up space may not be working as expected (for example, because the SAS tokens used for freeing up space have expired).

#### Your issue arises from using the storage emulator for development or test

You typically use the storage emulator during development and test to avoid the requirement for an Azure storage account. The common issues that can occur when you are using the storage emulator are:

- [Feature "X" is not working in the storage emulator](#)
- [Error "The value for one of the HTTP headers is not in the correct format" when using the storage emulator](#)
- [Running the storage emulator requires administrative privileges](#)

#### Feature "X" is not working in the storage emulator

The storage emulator does not support all of the features of the Azure storage services such as the file service. For more information, see [Use the Azure Storage Emulator for Development and Testing](#).

For those features that the storage emulator does not support, use the Azure storage service in the cloud.

#### Error "The value for one of the HTTP headers is not in the correct format" when using the storage emulator

You are testing your application that uses the Storage Client Library against the local storage emulator and method calls such as `CreateIfNotExists` fail with the error message "The value for one of the HTTP headers is

not in the correct format." This indicates that the version of the storage emulator you are using does not support the version of the storage client library you are using. The Storage Client Library adds the header **x-ms-version** to all the requests it makes. If the storage emulator does not recognize the value in the **x-ms-version** header, it rejects the request.

You can use the Storage Library Client logs to see the value of the **x-ms-version** header it is sending. You can also see the value of the **x-ms-version** header if you use Fiddler to trace the requests from your client application.

This scenario typically occurs if you install and use the latest version of the Storage Client Library without updating the storage emulator. You should either install the latest version of the storage emulator, or use cloud storage instead of the emulator for development and test.

#### **Running the storage emulator requires administrative privileges**

You are prompted for administrator credentials when you run the storage emulator. This only occurs when you are initializing the storage emulator for the first time. After you have initialized the storage emulator, you do not need administrative privileges to run it again.

For more information, see [Use the Azure Storage Emulator for Development and Testing](#). You can also initialize the storage emulator in Visual Studio, which will also require administrative privileges.

#### **You are encountering problems installing the Azure SDK for .NET**

When you try to install the SDK, it fails trying to install the storage emulator on your local machine. The installation log contains one of the following messages:

- CAQuietExec: Error: Unable to access SQL instance
- CAQuietExec: Error: Unable to create database

The cause is an issue with existing LocalDB installation. By default, the storage emulator uses LocalDB to persist data when it simulates the Azure storage services. You can reset your LocalDB instance by running the following commands in a command-prompt window before trying to install the SDK.

```
sqllocaldb stop v11.0
sqllocaldb delete v11.0
delete %USERPROFILE%\WAStorageEmulatorDb3.*
sqllocaldb create v11.0
```

The **delete** command removes any old database files from previous installations of the storage emulator.

#### **You have a different issue with a storage service**

If the previous troubleshooting sections do not include the issue you are having with a storage service, you should adopt the following approach to diagnosing and troubleshooting your issue.

- Check your metrics to see if there is any change from your expected base-line behavior. From the metrics, you may be able to determine whether the issue is transient or permanent, and which storage operations the issue is affecting.
- You can use the metrics information to help you search your server-side log data for more detailed information about any errors that are occurring. This information may help you troubleshoot and resolve the issue.
- If the information in the server-side logs is not sufficient to troubleshoot the issue successfully, you can use the Storage Client Library client-side logs to investigate the behavior of your client application, and tools such as Fiddler, Wireshark to investigate your network.

For more information about using Fiddler, see "[Appendix 1: Using Fiddler to capture HTTP and HTTPS traffic](#)."

For more information about using Wireshark, see "[Appendix 2: Using Wireshark to capture network traffic](#)."

## Appendices

The appendices describe several tools that you may find useful when you are diagnosing and troubleshooting issues with Azure Storage (and other services). These tools are not part of Azure Storage and some are third-party products. As such, the tools discussed in these appendices are not covered by any support agreement you may have with Microsoft Azure or Azure Storage, and therefore as part of your evaluation process you should examine the licensing and support options available from the providers of these tools.

### **Appendix 1: Using Fiddler to capture HTTP and HTTPS traffic**

Fiddler is a useful tool for analyzing the HTTP and HTTPS traffic between your client application and the Azure

storage service you are using.

**NOTE**

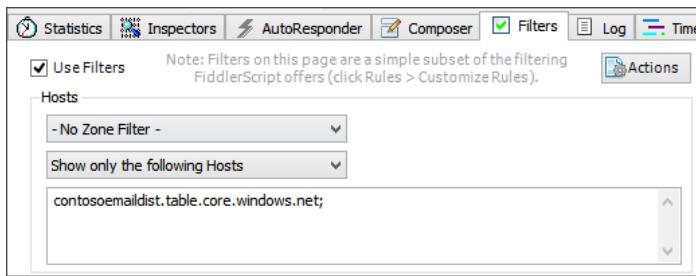
Fiddler can decode HTTPS traffic; you should read the Fiddler documentation carefully to understand how it does this, and to understand the security implications.

This appendix provides a brief walkthrough of how to configure Fiddler to capture traffic between the local machine where you have installed Fiddler and the Azure storage services.

After you have launched Fiddler, it will begin capturing HTTP and HTTPS traffic on your local machine. The following are some useful commands for controlling Fiddler:

- Stop and start capturing traffic. On the main menu, go to **File** and then click **Capture Traffic** to toggle capturing on and off.
- Save captured traffic data. On the main menu, go to **File**, click **Save**, and then click **All Sessions**: this enables you to save the traffic in a Session Archive file. You can reload a Session Archive later for analysis, or send it if requested to Microsoft support.

To limit the amount of traffic that Fiddler captures, you can use filters that you configure in the **Filters** tab. The following screenshot shows a filter that captures only traffic sent to the **contosoemaildist.table.core.windows.net** storage endpoint:

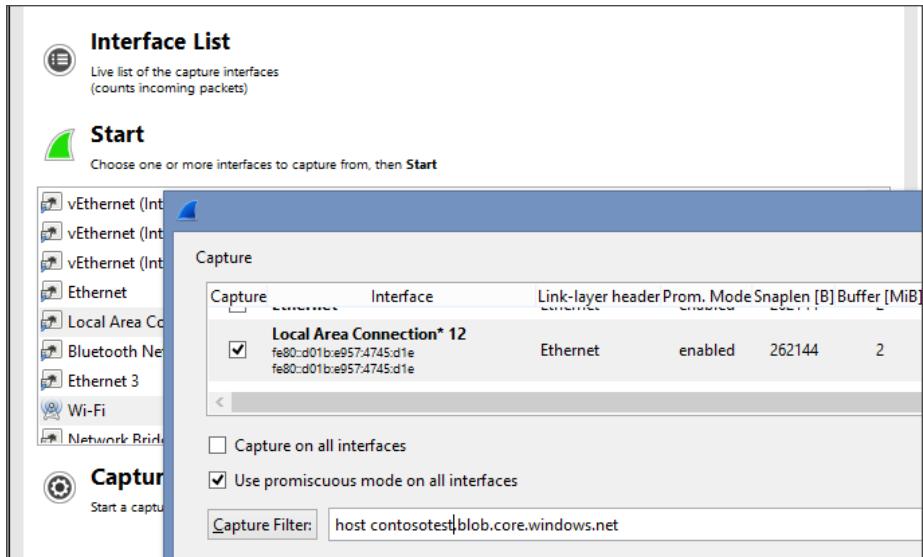


#### Appendix 2: Using Wireshark to capture network traffic

[Wireshark](#) is a network protocol analyzer that enables you to view detailed packet information for a wide range of network protocols.

The following procedure shows you how to capture detailed packet information for traffic from the local machine where you installed Wireshark to the table service in your Azure storage account.

1. Launch Wireshark on your local machine.
2. In the **Start** section, select the local network interface or interfaces that are connected to the internet.
3. Click **Capture Options**.
4. Add a filter to the **Capture Filter** textbox. For example, **host contosoemaildist.table.core.windows.net** will configure Wireshark to capture only packets sent to or from the table service endpoint in the **contosoemaildist** storage account. Check out the [complete list of Capture Filters](#).



5. Click **Start**. Wireshark will now capture all the packets send to or from the table service endpoint as you use your client application on your local machine.
6. When you have finished, on the main menu click **Capture** and then **Stop**.
7. To save the captured data in a Wireshark Capture File, on the main menu click **File** and then **Save**.

WireShark will highlight any errors that exist in the **packetlist** window. You can also use the **Expert Info** window (click **Analyze**, then **Expert Info**) to view a summary of errors and warnings.

Group	Protocol	Summary	Count
Checksum	Ethertype	Bad checksum	1
Malformed HTTP		Malformed Packet (Exception occurred)	1
Malformed TCP		New fragment overlaps old data (retransmission?)	1

You can also choose to view the TCP data as the application layer sees it by right-clicking on the TCP data and selecting **Follow TCP Stream**. This is useful if you captured your dump without a capture filter. For more information, see [Following TCP Streams](#).

**NOTE**

For more information about using Wireshark, see the [Wireshark Users Guide](#).

#### **Appendix 4: Using Excel to view metrics and log data**

Many tools enable you to download the Storage Metrics data from Azure table storage in a delimited format that makes it easy to load the data into Excel for viewing and analysis. Storage Logging data from Azure Blob Storage is already in a delimited format that you can load into Excel. However, you will need to add appropriate column headings based in the information at [Storage Analytics Log Format](#) and [Storage Analytics Metrics Table Schema](#).

To import your Storage Logging data into Excel after you download it from blob storage:

- On the **Data** menu, click **From Text**.
- Browse to the log file you want to view and click **Import**.
- On step 1 of the **Text Import Wizard**, select **Delimited**.

On step 1 of the **Text Import Wizard**, select **Semicolon** as the only delimiter and choose double-quote as the **Text qualifier**. Then click **Finish** and choose where to place the data in your workbook.

#### **Appendix 5: Monitoring with Application Insights for Azure DevOps**

You can also use the Application Insights feature for Azure DevOps as part of your performance and availability monitoring. This tool can:

- Make sure your web service is available and responsive. Whether your app is a web site or a device app that uses a web service, it can test your URL every few minutes from locations around the world, and let you know if there's a problem.
- Quickly diagnose any performance issues or exceptions in your web service. Find out if CPU or other resources are being stretched, get stack traces from exceptions, and easily search through log traces. If the app's performance drops below acceptable limits, Microsoft can send you an email. You can monitor both .NET and Java web services.

You can find more information at [What is Application Insights](#).

## Next steps

For more information about analytics in Azure Storage, see these resources:

- [Monitor a storage account in the Azure portal](#)
- [Storage analytics](#)
- [Storage analytics metrics](#)
- [Storage analytics metrics table schema](#)
- [Storage analytics logs](#)
- [Storage analytics log format](#)

# Troubleshoot latency using Storage Analytics logs

8/22/2022 • 3 minutes to read • [Edit Online](#)

Diagnosing and troubleshooting is a key skill for building and supporting client applications with Azure Storage.

Because of the distributed nature of an Azure application, diagnosing and troubleshooting both errors and performance issues may be more complex than in traditional environments.

The following steps demonstrate how to identify and troubleshoot latency issues using Azure Storage Analytic logs, and optimize the client application.

## Recommended steps

1. Download the [Storage Analytics logs](#).
2. Use the following PowerShell script to convert the raw format logs into tabular format:

```
$Columns =
(
 "version-number",
 "request-start-time",
 "operation-type",
 "request-status",
 "http-status-code",
 "end-to-end-latency-in-ms",
 "server-latency-in-ms",
 "authentication-type",
 "requester-account-name",
 "owner-account-name",
 "service-type",
 "request-url",
 "requested-object-key",
 "request-id-header",
 "operation-count",
 "requester-ip-address",
 "request-version-header",
 "request-header-size",
 "request-packet-size",
 "response-header-size",
 "response-packet-size",
 "request-content-length",
 "request-md5",
 "server-md5",
 "etag-identifier",
 "last-modified-time",
 "conditions-used",
 "user-agent-header",
 "referrer-header",
 "client-request-id"
)

$logs = Import-Csv "REPLACE THIS WITH FILE PATH" -Delimiter ";" -Header $Columns

$logs | Out-GridView -Title "Storage Analytic Log Parser"
```

3. The script will launch a GUI window where you can filter the information by columns, as shown below.

4. Narrow down the log entries based on "operation-type", and look for the log entry created during the issue's time frame.

version-number	request-start-time	operation-type	request-status	http-status-code	end-to-end-latency-in-ms	server-latency-in-ms	authentication-type	requester-account-name	owner-account-name	service-type	request-url
1.0	2019-10-07T19:17:53.7811547Z	GetBlobProperties	Success	200	12	12	authenticated	contoso\sa	contoso\sa	blob	https://contosoobj.blob.core.windows.net
1.0	2019-10-07T19:17:53.7811547Z	GetBlaBla	Success	200	106	26	authenticated	contoso\sa	contoso\sa	blob	https://contosoobj.blob.core.windows.net
1.0	2019-10-07T19:17:53.7846927Z	GetBlobProperties	Success	200	5	5	authenticated	contoso\sa	contoso\sa	blob	https://contosoobj.blob.core.windows.net
1.0	2019-10-07T19:17:53.7846927Z	GetBlaBlaProperties	Success	200	9	9	authenticated	contoso\sa	contoso\sa	blob	https://contosoobj.blob.core.windows.net
1.0	2019-10-07T19:17:53.7846927Z	GetSASWithKer	Success	200	8451	241	sas	contoso\sa	contoso\sa	blob	https://contosoobj.blob.core.windows.net
1.0	2019-10-07T19:21:25.848790Z	GetSASWithKer	Success	200	8457	515	sas	contoso\sa	contoso\sa	blob	https://contosoobj.blob.core.windows.net
1.0	2019-10-07T19:21:34.848917Z	GetSASWithKer	Success	200	10	10	sas	contoso\sa	contoso\sa	blob	https://contosoobj.blob.core.windows.net
1.0	2019-10-07T19:21:35.532195Z	GetBlaBla	SASNotAllowed	200	8453	452	sas	contoso\sa	contoso\sa	blob	https://contosoobj.blob.core.windows.net
1.0	2019-10-07T19:24:459941Z	GetBlaBlaProperties	SASNotAllowed	200	10	10	sas	contoso\sa	contoso\sa	blob	https://contosoobj.blob.core.windows.net
1.0	2019-10-07T19:21:35.531783Z	GetBlaBla	SASNotAllowed	200	8454	344	sas	contoso\sa	contoso\sa	blob	https://contosoobj.blob.core.windows.net

5. During the time when the issue occurred, the following values are important:

- Operation-type = GetBlob
  - request-status = SASNetworkError
  - End-to-End-Latency-In-Ms = 8453
  - Server-Latency-In-Ms = 391

End-to-End Latency is calculated using the following equation:

- End-to-End Latency = Server-Latency + Client Latency

Calculate the Client Latency using the log entry:

- Client Latency = End-to-End Latency – Server-Latency

Example:  $8453 - 391 = 8062\text{ms}$

The following table provides information about the high latency OperationType and RequestStatus results:

BLOB TYPE	REQUEST STATUS = SUCCESS	REQUEST STATUS = (SAS)NETWORKERROR	RECOMMENDATION
GetBlob	Yes	No	<a href="#">GetBlob Operation: RequestStatus = Success</a>
GetBlob	No	Yes	<a href="#">GetBlob Operation: RequestStatus = (SAS)NetworkError</a>

BLOB TYPE	REQUESTSTATUS= SUCCESS	REQUESTSTATUS= (SAS)NETWORKERROR	RECOMMENDATION
PutBlob	Yes	No	<a href="#">Put Operation: RequestStatus = Success</a>
PutBlob	No	Yes	<a href="#">Put Operation: RequestStatus = (SAS)NetworkError</a>

## Status results

### GetBlob Operation: RequestStatus = Success

Check the following values as mentioned in step 5 of the "Recommended steps" section:

- End-to-End Latency
- Server-Latency
- Client-Latency

In a **GetBlob Operation** with **RequestStatus = Success**, if **Max Time** is spent in **Client-Latency**, this indicates that Azure Storage is spending a large volume of time writing data to the client. This delay indicates a Client-Side Issue.

#### Recommendation:

- Investigate the code in your client.
- Use Wireshark, Microsoft Message Analyzer, or Tcping to investigate network connectivity issues from the client.

### GetBlob Operation: RequestStatus = (SAS)NetworkError

Check the following values as mentioned in step 5 of the "Recommended steps" section:

- End-to-End Latency
- Server-Latency
- Client-Latency

In a **GetBlob Operation** with **RequestStatus = (SAS)NetworkError**, if **Max Time** is spent in **Client-Latency**, the most common issue is that the client is disconnecting before a timeout expires in the storage service.

#### Recommendation:

- Investigate the code in your client to understand why and when the client disconnects from the storage service.
- Use Wireshark, Microsoft Message Analyzer, or Tcping to investigate network connectivity issues from the client.

### Put Operation: RequestStatus = Success

Check the following values as mentioned in step 5 of the "Recommended steps" section:

- End-to-End Latency
- Server-Latency
- Client-Latency

In a **Put Operation** with **RequestStatus = Success**, if **Max Time** is spent in **Client-Latency**, this indicates that the Client is taking more time to send data to the Azure Storage. This delay indicates a Client-Side Issue.

**Recommendation:**

- Investigate the code in your client.
- Use Wireshark, Microsoft Message Analyzer, or Tcping to investigate network connectivity issues from the client.

**Put Operation: RequestStatus = (SAS)NetworkError**

Check the following values as mentioned in step 5 of the "Recommended steps" section:

- End-to-End Latency
- Server-Latency
- Client-Latency

In a PutBlob Operation with RequestStatus = (SAS)NetworkError, if Max Time is spent in Client-Latency, the most common issue is that the client is disconnecting before a timeout expires in the storage service.

**Recommendation:**

- Investigate the code in your client to understand why and when the client disconnects from the storage service.
- Use Wireshark, Microsoft Message Analyzer, or Tcping to investigate network connectivity issues from the client.

# Azure Blob Storage API reference

8/22/2022 • 2 minutes to read • [Edit Online](#)

Find Blob Storage API reference, library packages, readme files, and getting started articles.

## .NET client libraries

The following table lists reference information for Blob Storage .NET APIs.

VERSION	REFERENCE DOCUMENTATION	PACKAGE	QUICKSTART
12.x	<a href="#">Azure Blob Storage client library for .NET</a>	<a href="#">Package (NuGet)</a>	<a href="#">Quickstart: Azure Blob Storage client library v12 for .NET</a>
11.x	<a href="#">Microsoft.Azure.Storage.Blob Namespace</a>	<a href="#">Package (NuGet)</a>	<a href="#">Quickstart: Azure Blob Storage client library v11 for .NET</a>

## Storage management .NET APIs

The following table lists reference information for Azure Storage management .NET APIs.

VERSION	REFERENCE DOCUMENTATION	PACKAGE
16.x	<a href="#">Microsoft.Azure.Management.Storage</a>	<a href="#">Package (NuGet)</a>

## Data movement .NET APIs

The following table lists reference information for Azure Storage data movement .NET APIs.

VERSION	REFERENCE DOCUMENTATION	PACKAGE
1.x	<a href="#">Data movement</a>	<a href="#">Package (NuGet)</a>

## Java client libraries

The following table lists reference information for Blob Storage Java APIs.

VERSION	REFERENCE DOCUMENTATION	PACKAGE	QUICKSTART
12.x	<a href="#">Azure Storage - Blobs</a>	<a href="#">Package (Maven)</a>	<a href="#">Quickstart: Manage blobs with Java v12 SDK</a>
8.x	<a href="#">com.microsoft.azure.storage.blob</a>	<a href="#">Package (Maven)</a>	<a href="#">Quickstart: Manage blobs with Java v8 SDK</a>

## Storage management Java APIs

The following table lists reference information for Azure Storage management Java APIs.

VERSION	REFERENCE DOCUMENTATION	PACKAGE
0.9.x	<a href="#">com.microsoft.azure.management.storage</a>	<a href="#">Package (Maven)</a>

## Python client libraries

The following table lists reference information for Blob Storage Python APIs.

VERSION	REFERENCE DOCUMENTATION	PACKAGE	QUICKSTART
12.x	<a href="#">Azure Storage client libraries v12 for Python</a>	<a href="#">Package (PyPI)</a>	<a href="#">Quickstart: Manage blobs with Python v12 SDK</a>
2.x	<a href="#">Azure Storage client libraries v2 for Python</a>	<a href="#">Package (PyPI)</a>	<a href="#">Quickstart: Manage blobs with Python v2.1 SDK</a>

## JavaScript client libraries

The following table lists reference information for Blob Storage JavaScript APIs.

VERSION	REFERENCE DOCUMENTATION	PACKAGE	QUICKSTART
12.x	<a href="#">Azure Storage Blob client library for JavaScript</a>	<a href="#">Package (npm)</a>	<a href="#">Quickstart: Manage blobs with JavaScript v12 SDK in Node.js</a> <a href="#">Quickstart: Manage blobs with JavaScript v12 SDK in a browser</a>
10.x	<a href="#">@azure/storage-blob</a>	<a href="#">Package (npm)</a>	<a href="#">Quickstart: Manage blobs with JavaScript v10 SDK in Node.js</a> <a href="#">Quickstart: Manage blobs with JavaScript v10 SDK in browser</a>

## C++ client libraries

The following table lists reference information for Blob Storage C++ APIs.

VERSION	REFERENCE DOCUMENTATION	SOURCE CODE/README	QUICKSTART
12.x	<a href="#">Azure SDK for C++ APIs</a>	<a href="#">Library source code</a>	<a href="#">Quickstart: Azure Blob Storage client library v12 for C++</a>

## REST APIs

The following table lists reference information for Blob Storage REST APIs.

REFERENCE DOCUMENTATION	OVERVIEW
<a href="#">Blob service REST API</a>	<a href="#">Blob service concepts</a>

## Other REST reference

[Azure Storage import-export REST API](#) helps you manage import/export jobs to transfer data to or from Blob Storage.

## Other languages and platforms

The following list contains links to libraries for other programming languages and platforms.

- [Ruby](#)
- [PHP](#)
- [iOS](#)
- [Android](#)

## PowerShell

[Azure PowerShell reference](#)

## Azure CLI

[Azure CLI reference](#)

# AzCopy v10 configuration settings (Azure Storage)

8/22/2022 • 5 minutes to read • [Edit Online](#)

AzCopy is a command-line utility that you can use to copy blobs or files to or from a storage account. This article contains a list of environment variables that you can use to configure AzCopy v10.

## NOTE

If you're looking for content to help you get started with AzCopy, see [Get started with AzCopy](#).

## AzCopy v10 environment variables

The following table describes each environment variable and provides links to content that can help you use the variable.

ENVIRONMENT VARIABLE	DESCRIPTION
AWS_ACCESS_KEY_ID	Amazon Web Services access key. Provides a key to authorize with Amazon Web Services. <a href="#">Copy data from Amazon S3 to Azure Storage by using AzCopy</a>
AWS_SECRET_ACCESS_KEY	Amazon Web Services secret access key. Provides a secret key to authorize with Amazon Web Services. <a href="#">Copy data from Amazon S3 to Azure Storage by using AzCopy</a>
AZCOPY_ACTIVE_DIRECTORY_ENDPOINT	The Azure Active Directory endpoint to use. This variable is only used for auto login, please use the command line flag instead when invoking the login command.
AZCOPY_AUTO_LOGIN_TYPE	Set this variable to <code>DEVICE</code> , <code>MSI</code> , or <code>SPN</code> . This variable provides the ability to authorize without using the <code>azcopy login</code> command. This mechanism is useful in cases where your operating system doesn't have a secret store such as a Linux <code>keyring</code> . See <a href="#">Authorize without a secret store</a> .
AZCOPY_BUFFER_GB	Specify the maximum amount of your system memory you want AzCopy to use when downloading and uploading files. Express this value in gigabytes (GB). See <a href="#">Optimize memory use</a>
AZCOPY_CACHE_PROXY_LOOKUP	By default AzCopy on Windows will cache proxy server lookups at hostname level (not taking URL path into account). Set to any other value than 'true' to disable the cache.
AZCOPY_CONCURRENCY_VALUE	Specifies the number of concurrent requests that can occur. You can use this variable to increase throughput. If your computer has fewer than 5 CPUs, then the value of this variable is set to <code>32</code> . Otherwise, the default value is equal to 16 multiplied by the number of CPUs. The maximum default value of this variable is <code>3000</code> , but you can manually set this value higher or lower. See <a href="#">Increase concurrency</a>

ENVIRONMENT VARIABLE	DESCRIPTION
AZCOPY_CONCURRENT_FILES	Overrides the (approximate) number of files that are in progress at any one time, by controlling how many files we concurrently initiate transfers for.
AZCOPY_CONCURRENT_SCAN	Controls the (max) degree of parallelism used during scanning. Only affects parallelized enumerators, which include Azure Files/Blobs, and local file systems.
AZCOPY_CONTENT_TYPE_MAP	<p>Overrides one or more of the default MIME type mappings defined by your operating system. Set this variable to the path of a JSON file that defines any mapping. Here's the contents of an example JSON file:</p> <pre>{   "MIMETypeMapping": {     ".323": "text/h323",     ".aaf": "application/octet-stream",     ".aca": "application/octet-stream",     ".accdb": "application/msaccess",   } }</pre>
AZCOPY_DEFAULT_SERVICE_API_VERSION	Overrides the service API version so that AzCopy could accommodate custom environments such as Azure Stack.
AZCOPY_DISABLE_HIERARCHICAL_SCAN	Applies only when Azure Blobs is the source. Concurrent scanning is faster but employs the hierarchical listing API, which can result in more IOs/cost. Specify 'true' to sacrifice performance but save on cost.
AZCOPY_JOB_PLAN_LOCATION	Overrides where the job plan files (used for progress tracking and resuming) are stored, to avoid filling up a disk.
AZCOPY_LOG_LOCATION	Overrides where the log files are stored, to avoid filling up a disk.
AZCOPY_MSI_CLIENT_ID	The client ID of a user-assigned managed identity. Use when <code>AZCOPY_AUTO_LOGIN_TYPE</code> is set to <code>MSI</code> . See <a href="#">Authorize without a secret store</a>
AZCOPY_MSI_OBJECT_ID	The object ID of the user-assigned managed identity. Use when <code>AZCOPY_AUTO_LOGIN_TYPE</code> is set to <code>MSI</code> . See <a href="#">Authorize without a secret store</a>
AZCOPY_MSI_RESOURCE_STRING	The resource ID of the user-assigned managed identity. See <a href="#">Authorize without a secret store</a>
AZCOPY_PACE_PAGE_BLOBS	Should throughput for page blobs automatically be adjusted to match Service limits? Default is true. Set to 'false' to disable

ENVIRONMENT VARIABLE	DESCRIPTION
AZCOPY_PARALLEL_STAT_FILES	Causes AzCopy to look up file properties on parallel 'threads' when scanning the local file system. The threads are drawn from the pool defined by AZCOPY_CONCURRENT_SCAN. Setting this to true may improve scanning performance on Linux. Not needed or recommended on Windows.
AZCOPY_SHOW_PERF_STATES	If set, to anything, on-screen output will include counts of chunks by state
AZCOPY_SPA_APPLICATION_ID	The application ID of your service principal's app registration. Use when <code>AZCOPY_AUTO_LOGIN_TYPE</code> is set to <code>SPN</code> . See <a href="#">Authorize without a secret store</a>
AZCOPY_SPA_CERT_PASSWORD	The password of a certificate. Use when <code>AZCOPY_AUTO_LOGIN_TYPE</code> is set to <code>SPN</code> . See <a href="#">Authorize without a secret store</a>
AZCOPY_SPA_CERT_PATH	The relative or fully qualified path to a certificate file. Use when <code>AZCOPY_AUTO_LOGIN_TYPE</code> is set to <code>SPN</code> . See <a href="#">Authorize without a secret store</a>
AZCOPY_SPA_CLIENT_SECRET	The client secret. Use when <code>AZCOPY_AUTO_LOGIN_TYPE</code> is set to <code>SPN</code> . See <a href="#">Authorize without a secret store</a>
AZCOPY_TENANT_ID	The Azure Active Directory tenant ID to use for OAuth device interactive login. This variable is only used for auto login, please use the command line flag instead when invoking the login command.
AZCOPY_TUNE_TO_CPU	Set to false to prevent AzCopy from taking CPU usage into account when autotuning its concurrency level (for example, in the benchmark command).
AZCOPY_USER_AGENT_PREFIX	Add a prefix to the default AzCopy User Agent, which is used for telemetry purposes. A space is automatically inserted.
GOOGLE_APPLICATION_CREDENTIALS	The absolute path to the service account key file Provides a key to authorize with Google Cloud Storage. <a href="#">Copy data from Google Cloud Storage to Azure Storage by using AzCopy (preview)</a>
HTTPS_PROXY	Configures proxy settings for AzCopy. Set this variable to the proxy IP address and proxy port number. For example, <code>xx.xxx.xx.xxx:xx</code> . If you run AzCopy on Windows, AzCopy automatically detects proxy settings, so you don't have to use this setting in Windows. If you choose to use this setting in Windows, it will override automatic detection. See <a href="#">Configure proxy settings</a>

## Configure proxy settings

To configure the proxy settings for AzCopy, set the `HTTPS_PROXY` environment variable. If you run AzCopy on Windows, AzCopy automatically detects proxy settings, so you don't have to use this setting in Windows. If you choose to use this setting in Windows, it will override automatic detection.

OPERATING SYSTEM	COMMAND
Windows	In a command prompt use: <pre>set HTTPS_PROXY=&lt;proxy IP&gt;:&lt;proxy port&gt;</pre> In PowerShell use: <pre>\$env:HTTPS_PROXY=&lt;proxy IP&gt;:&lt;proxy port&gt;"</pre>
Linux	<pre>export HTTPS_PROXY=&lt;proxy IP&gt;:&lt;proxy port&gt;</pre>
macOS	<pre>export HTTPS_PROXY=&lt;proxy IP&gt;:&lt;proxy port&gt;</pre>

Currently, AzCopy doesn't support proxies that require authentication with NTLM or Kerberos.

### Bypassing a proxy

If you are running AzCopy on Windows, and you want to tell it to use *no proxy* at all (instead of auto-detecting the settings) use these commands. With these settings, AzCopy will not look up or attempt to use any proxy.

OPERATING SYSTEM	ENVIRONMENT	COMMANDS
Windows	Command prompt (CMD)	<pre>set HTTPS_PROXY=dummy.invalid</pre> <pre>set NO_PROXY=*</pre>
Windows	PowerShell	<pre>\$env:HTTPS_PROXY="dummy.invalid"</pre> <pre>\$env:NO_PROXY="*"</pre>

On other operating systems, simply leave the HTTPS\_PROXY variable unset if you want to use no proxy.

## See also

- [Get started with AzCopy](#)
- [Optimize the performance of AzCopy v10 with Azure Storage](#)
- [Troubleshoot AzCopy V10 issues in Azure Storage by using log files](#)

# azcopy

8/22/2022 • 2 minutes to read • [Edit Online](#)

Current version: 10.15.0

AzCopy is a command-line tool that moves data into and out of Azure Storage. See the [Get started with AzCopy](#) article to download AzCopy and learn about the ways that you can provide authorization credentials to the storage service.

## Synopsis

The general format of the commands is: `azcopy [command] [arguments] --[flag-name]=[flag-value]`.

To report issues or to learn more about the tool, see <https://github.com/Azure/azure-storage-azcopy>.

## Related conceptual articles

- [Get started with AzCopy](#)
- [Tutorial: Migrate on-premises data to cloud storage with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Options

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it is omitted, the throughput isn't capped.

`-h` , `--help` help for azcopy

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is `'.core.windows.net;core.chinacloudapi.cn;core.cloudapi.de;core.usgovcloudapi.net;*.storage.azure.net'`. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [Get started with AzCopy](#)
- [azcopy bench](#)
- [azcopy copy](#)
- [azcopy doc](#)
- [azcopy env](#)
- [azcopy jobs](#)
- [azcopy jobs clean](#)
- [azcopy jobs list](#)
- [azcopy jobs remove](#)
- [azcopy jobs resume](#)

- [azcopy jobs show](#)
- [azcopy list](#)
- [azcopy login](#)
- [azcopy login status](#)
- [azcopy logout](#)
- [azcopy make](#)
- [azcopy remove](#)
- [azcopy sync](#)
- [azcopy set-properties](#)

# azcopy bench

8/22/2022 • 4 minutes to read • [Edit Online](#)

Runs a performance benchmark by uploading or downloading test data to or from a specified destination. For uploads, the test data is automatically generated.

The benchmark command runs the same process as 'copy', except that:

- Instead of requiring both source and destination parameters, benchmark takes just one. This is the blob container, Azure Files Share, or Azure Data Lake Storage Gen2 file system that you want to upload to or download from.
- The 'mode' parameter describes whether AzCopy should test uploads to or downloads from given target. Valid values are 'Upload' and 'Download'. Default value is 'Upload'.
- For upload benchmarks, the payload is described by command line parameters, which control how many files are auto-generated and how big they are. The generation process takes place entirely in memory. Disk isn't used.
- For downloads, the payload consists of whichever files already exist at the source. (See example below about how to generate test files if needed).
- Only a few of the optional parameters that are available to the copy command are supported.
- Additional diagnostics are measured and reported.
- For uploads, the default behavior is to delete the transferred data at the end of the test run. For downloads, the data is never actually saved locally.

Benchmark mode will automatically tune itself to the number of parallel TCP connections that gives the maximum throughput. It will display that number at the end. To prevent auto-tuning, set the COPY\_CONCURRENCY\_VALUE environment variable to a specific number of connections.

All the usual authentication types are supported. However, the most convenient approach for benchmarking upload is typically to create an empty container with a SAS token and use SAS authentication. (Download mode requires a set of test data to be present in the target container.)

```
azcopy bench [destination] [flags]
```

## Examples

Run an upload benchmark with default parameters (suitable for benchmarking networks up to 1 Gbps):

```
azcopy bench "https://[account].blob.core.windows.net/[container]?<SAS>"
```

Run a benchmark test that uploads 100 files, each 2 GiB in size: (suitable for benchmarking on a fast network, e.g. 10 Gbps):

```
azcopy bench "https://[account].blob.core.windows.net/[container]?<SAS>" --file-count 100 --size-per-file 2G
```

Same as above, but use 50,000 files, each 8 MiB in size and compute their MD5 hashes (in the same way that the --put-md5 flag does this in the copy command). The purpose of --put-md5 when benchmarking is to test whether MD5 computation affects throughput for the selected file count and size:

```
azcopy bench --mode='Upload' "https://[account].blob.core.windows.net/[container]?<SAS>" --file-count 50000 -
-size-per-file 8M --put-md5
```

Run a benchmark test that downloads existing files from a target

```
azcopy bench --mode='Download' "https://[account].blob.core.windows.net/[container]?<SAS?!"
```

Run an upload that doesn't delete the transferred files. (These files can then serve as the payload for a download test)

```
azcopy bench "https://[account].blob.core.windows.net/[container]?<SAS>" --file-count 100 --delete-test-data=false
```

## Options

`--blob-type string` defines the type of blob at the destination. Used to allow benchmarking different blob types. Identical to the same-named parameter in the copy command (default "Detect")

`--block-size-mb float` Use this block size (specified in MiB). Default is automatically calculated based on file size. Decimal fractions are allowed - for example, 0.25. Identical to the same-named parameter in the copy command

`--check-length` Check the length of a file on the destination after the transfer. If there's a mismatch between source and destination, the transfer is marked as failed. (default true)

`--delete-test-data` If true, the benchmark data will be deleted at the end of the benchmark run. Set it to false if you want to keep the data at the destination - for example, to use it for manual tests outside benchmark mode (default true)

`--file-count` (uint) number of auto-generated data files to use (default 100)

`-h , --help` help for bench

`--log-level` (string) define the log verbosity for the log file, available levels: INFO(all requests/responses), WARNING(slow responses), ERROR(only failed requests), and NONE(no output logs). (default "INFO")

`--mode` (string) Defines if Azcopy should test uploads or downloads from this target. Valid values are 'upload' and 'download'. Defaulted option is 'upload'. (default "upload")

`--number-of-folders` (uint) If larger than 0, create folders to divide up the data.

`--put-md5` Create an MD5 hash of each file, and save the hash as the Content-MD5 property of the destination blob/file. (By default the hash is NOT created.) Identical to the same-named parameter in the copy command

`--size-per-file` (string) Size of each auto-generated data file. Must be a number immediately followed by K, M or G. E.g. 12k or 200G (default "250M")

## Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it's omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)

# azcopy copy

8/22/2022 • 15 minutes to read • [Edit Online](#)

Copies source data to a destination location.

## Synopsis

Copies source data to a destination location. The supported directions are:

- local <-> Azure Blob (SAS or OAuth authentication)
- local <-> Azure Files (Share/directory SAS authentication)
- local <-> Azure Data Lake Storage Gen2 (SAS, OAuth, or SharedKey authentication)
- Azure Blob (SAS or public) -> Azure Blob (SAS or OAuth authentication)
- Azure Blob (SAS or public) -> Azure Files (SAS)
- Azure Files (SAS) -> Azure Files (SAS)
- Azure Files (SAS) -> Azure Blob (SAS or OAuth authentication)
- AWS S3 (Access Key) -> Azure Block Blob (SAS or OAuth authentication)
- Google Cloud Storage (Service Account Key) -> Azure Block Blob (SAS or OAuth authentication)

Refer to the examples for more information.

## Advanced

AzCopy automatically detects the content type of the files when uploading from the local disk, based on the file extension or content (if no extension is specified).

The built-in lookup table is small, but on Unix, it's augmented by the local system's mime.types file(s) if available under one or more of these names:

- /etc/mime.types
- /etc/apache2/mime.types
- /etc/apache/mime.types

On Windows, MIME types are extracted from the registry. This feature can be turned off with the help of a flag. Refer to the flag section.

If you set an environment variable by using the command line, that variable will be readable in your command line history. Consider clearing variables that contain credentials from your command line history. To keep variables from appearing in your history, you can use a script to prompt the user for their credentials, and to set the environment variable.

```
azcopy copy [source] [destination] [flags]
```

## Examples

Upload a single file by using OAuth authentication. If you haven't yet logged into AzCopy, please run the azcopy login command before you run the following command.

```
azcopy cp "/path/to/file.txt" "https://[account].blob.core.windows.net/[container]/[path/to/blob]"
```

Same as above, but this time also compute MD5 hash of the file content and save it as the blob's Content-MD5

property:

```
azcopy cp "/path/to/file.txt" "https://[account].blob.core.windows.net/[container]/[path/to/blob]" --put-md5
```

Upload a single file by using a SAS token:

```
azcopy cp "/path/to/file.txt" "https://[account].blob.core.windows.net/[container]/[path/to/blob]?[SAS]"
```

Upload a single file by using a SAS token and piping (block blobs only):

```
cat "/path/to/file.txt" | azcopy cp "https://[account].blob.core.windows.net/[container]/[path/to/blob]?[SAS]" --from-to PipeBlob
```

Upload a single file by using OAuth and piping (block blobs only):

```
cat "/path/to/file.txt" | azcopy cp "https://[account].blob.core.windows.net/[container]/[path/to/blob]" --from-to PipeBlob
```

Upload an entire directory by using a SAS token:

```
azcopy cp "/path/to/dir" "https://[account].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" --recursive=true
```

or

```
azcopy cp "/path/to/dir" "https://[account].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" --recursive=true --put-md5
```

Upload a set of files by using a SAS token and wildcard (\*) characters:

```
azcopy cp "/path/*foo/*bar/*.pdf" "https://[account].blob.core.windows.net/[container]/[path/to/directory]?[SAS]"
```

Upload files and directories by using a SAS token and wildcard (\*) characters:

```
azcopy cp "/path/*foo/*bar*" "https://[account].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" --recursive=true
```

Upload files and directories to Azure Storage account and set the query-string encoded tags on the blob.

- To set tags {key = "bla bla", val = "foo"} and {key = "bla bla 2", val = "bar"}, use the following syntax:

```
azcopy cp "/path/*foo/*bar*" "https://[account].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" --blob-tags="bla%20bla=foo&bla%20bla%202=bar"
```

- Keys and values are URL encoded and the key-value pairs are separated by an ampersand(&)
- While setting tags on the blobs, there are additional permissions('t' for tags) in SAS without which the service will give authorization error back.

Download a single file by using OAuth authentication. If you haven't yet logged into AzCopy, please run the azcopy login command before you run the following command.

```
azcopy cp "https://[account].blob.core.windows.net/[container]/[path/to/blob]" "/path/to/file.txt"
```

Download a single file by using a SAS token:

```
azcopy cp "https://[account].blob.core.windows.net/[container]/[path/to/blob]?[SAS]" "/path/to/file.txt"
```

Download a single file by using a SAS token and then piping the output to a file (block blobs only):

```
azcopy cp "https://[account].blob.core.windows.net/[container]/[path/to/blob]?[SAS]" --from-to BlobPipe > "/path/to/file.txt"
```

Download a single file by using OAuth and then piping the output to a file (block blobs only):

```
azcopy cp "https://[account].blob.core.windows.net/[container]/[path/to/blob]" --from-to BlobPipe > "/path/to/file.txt"
```

Download an entire directory by using a SAS token:

```
azcopy cp "https://[account].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" "/path/to/dir" --recursive=true
```

A note about using a wildcard character (\*) in URLs:

There's only two supported ways to use a wildcard character in a URL.

- You can use one just after the final forward slash (/) of a URL. This copies all of the files in a directory directly to the destination without placing them into a subdirectory.
- You can also use one in the name of a container as long as the URL refers only to a container and not to a blob. You can use this approach to obtain files from a subset of containers.

Download the contents of a directory without copying the containing directory itself.

```
azcopy cp "https://[srcaccount].blob.core.windows.net/[container]/[path/to/folder]/*?[SAS]" "/path/to/dir"
```

Download an entire storage account.

```
azcopy cp "https://[srcaccount].blob.core.windows.net/" "/path/to/dir" --recursive
```

Download a subset of containers within a storage account by using a wildcard symbol (\*) in the container name.

```
azcopy cp "https://[srcaccount].blob.core.windows.net/[container*name]" "/path/to/dir" --recursive
```

Download all the versions of a blob from Azure Storage to local directory. Ensure that source is a valid blob, destination is a local folder and `versionidsFile` which takes in a path to the file where each version is written on a separate line. All the specified versions will get downloaded in the destination folder specified.

```
azcopy cp "https://[srcaccount].blob.core.windows.net/[containername]/[blobname]" "/path/to/dir" --list-of-versions="/another/path/to/dir/[versionidsFile]"
```

Copy a single blob to another blob by using a SAS token.

```
azcopy cp "https://[srcaccount].blob.core.windows.net/[container]/[path/to/blob]?[SAS]" "https://[destaccount].blob.core.windows.net/[container]/[path/to/blob]?[SAS]"
```

Copy a single blob to another blob by using a SAS token and an OAuth token.

```
azcopy cp "https://[srcaccount].blob.core.windows.net/[container]/[path/to/blob]" "https://[destaccount].blob.core.windows.net/[container]/[path/to/blob]"
```

Copy one blob virtual directory to another by using a SAS token:

```
azcopy cp "https://[srcaccount].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" "https://[destaccount].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" --recursive=true
```

Copy all blob containers, directories, and blobs from storage account to another by using a SAS token:

```
azcopy cp "https://[srcaccount].blob.core.windows.net?[SAS]" "https://[destaccount].blob.core.windows.net?[SAS]" --recursive=true
```

Copy a single object to Blob Storage from Amazon Web Services (AWS) S3 by using an access key and a SAS token. First, set the environment variable AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY for AWS S3 source.

```
azcopy cp "https://s3.amazonaws.com/[bucket]/[object]" "https://[destaccount].blob.core.windows.net/[container]/[path/to/blob]?[SAS]"
```

Copy an entire directory to Blob Storage from AWS S3 by using an access key and a SAS token. First, set the environment variable AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY for AWS S3 source.

```
azcopy cp "https://s3.amazonaws.com/[bucket]/[folder]" "https://[destaccount].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" --recursive=true
```

Refer to <https://docs.aws.amazon.com/AmazonS3/latest/user-guide/using-folders.html> to better understand the

[folder] placeholder.

Copy all buckets to Blob Storage from Amazon Web Services (AWS) by using an access key and a SAS token. First, set the environment variable AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY for AWS S3 source.

```
azcopy cp "https://s3.amazonaws.com/" "https://[destaccount].blob.core.windows.net?[SAS]" --recursive=true
```

Copy all buckets to Blob Storage from an Amazon Web Services (AWS) region by using an access key and a SAS token. First, set the environment variable AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY for AWS S3 source.

```
azcopy cp "https://s3-[region].amazonaws.com/" "https://[destaccount].blob.core.windows.net?[SAS]" --recursive=true
```

Copy a subset of buckets by using a wildcard symbol (\*) in the bucket name. Like the previous examples, you'll need an access key and a SAS token. Make sure to set the environment variable AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY for AWS S3 source.

```
azcopy cp "https://s3.amazonaws.com/[bucket*name]/" "https://[destaccount].blob.core.windows.net?[SAS]" --recursive=true
```

Copy blobs from one blob storage to another and preserve the tags from source. To preserve tags, use the following syntax:

```
azcopy cp "https://[account].blob.core.windows.net/[source_container]/[path/to/directory]?[SAS]"
"https://[account].blob.core.windows.net/[destination_container]/[path/to/directory]?[SAS]" --s2s-preserve-blob-tags=true
```

Transfer files and directories to Azure Storage account and set the given query-string encoded tags on the blob.

- To set tags {key = "bla bla", val = "foo"} and {key = "bla bla 2", val = "bar"}, use the following syntax:

```
azcopy cp "https://[account].blob.core.windows.net/[source_container]/[path/to/directory]?[SAS]"
"https://[account].blob.core.windows.net/[destination_container]/[path/to/directory]?[SAS]" --blob-tags="bla%20bla=foo&bla%20bla%202=bar"
```

- Keys and values are URL encoded and the key-value pairs are separated by an ampersand(&)
- While setting tags on the blobs, there are additional permissions('t' for tags) in SAS without which the service will give authorization error back.

Copy a single object to Blob Storage from Google Cloud Storage (GCS) by using a service account key and a SAS token. First, set the environment variable GOOGLE\_APPLICATION\_CREDENTIALS for GCS source.

```
azcopy cp "https://storage.cloud.google.com/[bucket]/[object]"
"https://[destaccount].blob.core.windows.net/[container]/[path/to/blob]?[SAS]"
```

Copy an entire directory to Blob Storage from Google Cloud Storage (GCS) by using a service account key and a SAS token. First, set the environment variable GOOGLE\_APPLICATION\_CREDENTIALS for GCS source.

```
azcopy cp "https://storage.cloud.google.com/[bucket]/[folder]"
"https://[destaccount].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" --recursive=true
```

Copy an entire bucket to Blob Storage from Google Cloud Storage (GCS) by using a service account key and a SAS token. First, set the environment variable GOOGLE\_APPLICATION\_CREDENTIALS for GCS source.

```
azcopy cp "https://storage.cloud.google.com/[bucket]" "https://[destaccount].blob.core.windows.net/?[SAS]" --recursive=true
```

Copy all buckets to Blob Storage from Google Cloud Storage (GCS) by using a service account key and a SAS token. First, set the environment variables GOOGLE\_APPLICATION\_CREDENTIALS and GOOGLE\_CLOUD\_PROJECT=<project-id> for GCS source

```
azcopy cp "https://storage.cloud.google.com/" "https://[destaccount].blob.core.windows.net/?[SAS]" --recursive=true
```

Copy a subset of buckets by using a wildcard symbol (\*) in the bucket name from Google Cloud Storage (GCS) by using a service account key and a SAS token for destination. First, set the environment variables

```
GOOGLE_APPLICATION_CREDENTIALS and GOOGLE_CLOUD_PROJECT=<project-id> for GCS source
```

```
azcopy cp "https://storage.cloud.google.com/[bucket*name]/" "https://[destaccount].blob.core.windows.net/?[SAS]" --recursive=true
```

## Options

**--as-subdir** True by default. Places folder sources as subdirectories under the destination. (default true)

**--backup** Activates Windows' SeBackupPrivilege for uploads, or SeRestorePrivilege for downloads, to allow AzCopy to see read all files, regardless of their file system permissions, and to restore all permissions. Requires that the account running AzCopy already has these permissions (for example, has Administrator rights or is a member of the 'Backup Operators' group). This flag activates privileges that the account already has

**--blob-tags** (string) Set tags on blobs to categorize data in your storage account

**--blob-type** (string) Defines the type of blob at the destination. This is used for uploading blobs and when copying between accounts (default 'Detect'). Valid values include 'Detect', 'BlockBlob', 'PageBlob', and 'AppendBlob'. When copying between accounts, a value of 'Detect' causes AzCopy to use the type of source blob to determine the type of the destination blob. When uploading a file, 'Detect' determines if the file is a VHD or a VHDX file based on the file extension. If the file is either a VHD or VHDX file, AzCopy treats the file as a page blob. (default "Detect")

**--block-blob-tier** (string) upload block blob to Azure Storage using this blob tier. (default "None")

**--block-size-mb** (float) Use this block size (specified in MiB) when uploading to Azure Storage, and downloading from Azure Storage. The default value is automatically calculated based on file size. Decimal fractions are allowed (For example: 0.25).

**--cache-control** (string) Set the cache-control header. Returned on download.

**--check-length** Check the length of a file on the destination after the transfer. If there's a mismatch between source and destination, the transfer is marked as failed. (default true)

**--check-md5** (string) Specifies how strictly MD5 hashes should be validated when downloading. Only available when downloading. Available options: NoCheck, LogOnly, FailIfDifferent, FailIfDifferentOrMissing. (default 'FailIfDifferent') (default "FailIfDifferent")

**--content-disposition** (string) Set the content-disposition header. Returned on download.

**--content-encoding** (string) Set the content-encoding header. Returned on download.

**--content-language** (string) Set the content-language header. Returned on download.

**--content-type** (string) Specifies the content type of the file. Implies no-guess-mime-type. Returned on download.

**--cpk-by-name** (string) Client provided key by name that lets clients making requests against Azure Blob storage an option to provide an encryption key on a per-request basis. Provided key name will be fetched from Azure Key Vault and will be used to encrypt the data

**--cpk-by-value** Client provided key by name that let clients making requests against Azure Blob storage an option to provide an encryption key on a per-request basis. Provided key and its hash will be fetched from environment variables

**--decompress** Automatically decompress files when downloading, if their content-encoding indicates that they're compressed. The supported content-encoding values are 'gzip' and 'deflate'. File extensions of '.gz'/.gzip'

or '.zz' aren't necessary, but will be removed if present.

**--disable-auto-decoding** False by default to enable automatic decoding of illegal chars on Windows. Can be set to true to disable automatic decoding.

**--dry-run** Prints the file paths that would be copied by this command. This flag doesn't copy the actual files.

**--exclude-attributes** (string) (Windows only) Exclude files whose attributes match the attribute list. For example: A;S;R

**--exclude-blob-type** (string) Optionally specifies the type of blob (BlockBlob/ PageBlob/ AppendBlob) to exclude when copying blobs from the container or the account. Use of this flag isn't applicable for copying data from non azure-service to service. More than one blob should be separated by ':'.

**--exclude-path** (string) Exclude these paths when copying. This option doesn't support wildcard characters (\*). Checks relative path prefix(For example: myFolder;myFolder/subDirName/file.pdf). When used in combination with account traversal, paths don't include the container name.

**--exclude-pattern** (string) Exclude these files when copying. This option supports wildcard characters (\*)

**--exclude-regex** (string) Exclude all the relative path of the files that align with regular expressions. Separate regular expressions with ':'.

**--follow-symlinks** Follow symbolic links when uploading from local file system.

**--force-if-read-only** When overwriting an existing file on Windows or Azure Files, force the overwrite to work even if the existing file has its read-only attribute set

**--from-to** (string) Optionally specifies the source destination combination. For Example: LocalBlob, BlobLocal, LocalBlobFS. Piping: BlobPipe, PipeBlob

**-h , --help** help for copy

**--include-after** (string) Include only those files modified on or after the given date/time. The value should be in ISO8601 format. If no timezone is specified, the value is assumed to be in the local timezone of the machine running AzCopy. E.g., `2020-08-19T15:04:00Z` for a UTC time, or `2020-08-19` for midnight (00:00) in the local timezone. As of AzCopy 10.5, this flag applies only to files, not folders, so folder properties won't be copied when using this flag with `--preserve-smb-info` or `--preserve-smb-permissions`.

**--include-attributes** (string) (Windows only) Include files whose attributes match the attribute list. For example: A;S;R

**--include-before** (string) Include only those files modified before or on the given date/time. The value should be in ISO8601 format. If no timezone is specified, the value is assumed to be in the local timezone of the machine running AzCopy. for example, `2020-08-19T15:04:00Z` for a UTC time, or `2020-08-19` for midnight (00:00) in the local timezone. As of AzCopy 10.7, this flag applies only to files, not folders, so folder properties won't be copied when using this flag with `--preserve-smb-info` or `--preserve-smb-permissions`.

**--include-directory-stub** False by default to ignore directory stubs. Directory stubs are blobs with metadata `hdi_isfolder:true`. Setting value to true will preserve directory stubs during transfers.

**--include-path** (string) Include only these paths when copying. This option doesn't support wildcard characters (\*). Checks relative path prefix (For example: myFolder;myFolder/subDirName/file.pdf).

**--include-pattern** (string) Include only these files when copying. This option supports wildcard characters (\*). Separate files by using a ':'.

**--include-regex** (string) Include only the relative path of the files that align with regular expressions. Separate regular expressions with ':'.

**--list-of-versions** (string) Specifies a file where each version ID is listed on a separate line. Ensure that the source must point to a single blob and all the version IDs specified in the file using this flag must belong to the source blob only. AzCopy will download the specified versions in the destination folder provided.

**--log-level** (string) Define the log verbosity for the log file, available levels: INFO(all requests/responses), WARNING(slow responses), ERROR(only failed requests), and NONE(no output logs). (default 'INFO'). (default "INFO")

**--metadata** (string) Upload to Azure Storage with these key-value pairs as metadata.

**--no-guess-mime-type** Prevents AzCopy from detecting the content-type based on the extension or content of the file.

**--overwrite** (string) Overwrite the conflicting files and blobs at the destination if this flag is set to true. (default 'true') Possible values include 'true', 'false', 'prompt', and 'ifSourceNewer'. For destinations that support folders, conflicting folder-level properties will be overwritten this flag is 'true' or if a positive response is provided to the prompt. (default "true")

**--page-blob-tier** (string) Upload page blob to Azure Storage using this blob tier. (default 'None'). (default "None")

**--preserve-last-modified-time** Only available when destination is file system.

**--preserve-owner** Only has an effect in downloads, and only when **--preserve-smb-permissions** is used. If true (the default), the file Owner and Group are preserved in downloads. If set to false,

**--preserve-smb-permissions** will still preserve ACLs but Owner and Group will be based on the user running AzCopy (default true)

**--preserve-permissions** False by default. Preserves ACLs between aware resources (Windows and Azure Files, or Azure Data Lake Storage Gen2 to Azure Data Lake Storage Gen2). For Hierarchical Namespace accounts, you'll need a container SAS or OAuth token with Modify Ownership and Modify Permissions permissions. For downloads, you'll also need the **--backup** flag to restore permissions where the new Owner won't be the user running AzCopy. This flag applies to both files and folders, unless a file-only filter is specified (for example, include-pattern).

**--preserve-smb-info** For SMB-aware locations, flag will be set to true by default. Preserves SMB property info (last write time, creation time, attribute bits) between SMB-aware resources (Windows and Azure Files). Only the attribute bits supported by Azure Files will be transferred; any others will be ignored. This flag applies to both files and folders, unless a file-only filter is specified (for example, include-pattern). The info transferred for folders is the same as that for files, except for **Last Write Time** which is never preserved for folders. (default true)

**--put-md5** Create an MD5 hash of each file, and save the hash as the Content-MD5 property of the destination blob or file. (By default the hash is NOT created.) Only available when uploading.

**--recursive** Look into subdirectories recursively when uploading from local file system.

**--s2s-detect-source-changed** Detect if the source file/blob changes while it is being read. (This parameter only applies to service-to-service copies, because the corresponding check is permanently enabled for uploads and downloads.)

**--s2s-handle-invalid-metadata** (string) Specifies how invalid metadata keys are handled. Available options: ExcludelfInvalid, FaillfInvalid, RenamelfInvalid. (default 'ExcludelfInvalid'). (default "ExcludelfInvalid")

**--s2s-preserve-access-tier** Preserve access tier during service to service copy. Refer to [Azure Blob storage: hot, cool, and archive access tiers](#) to ensure destination storage account supports setting access tier. In the cases that setting access tier isn't supported, make sure to use s2sPreserveAccessTier=false to bypass copying access tier.

(default true). (default true)

`--s2s-preserve-blob-tags` Preserve index tags during service to service transfer from one blob storage to another

`--s2s-preserve-properties` Preserve full properties during service to service copy. For AWS S3 and Azure File non-single file source, the list operation doesn't return full properties of objects and files. To preserve full properties, AzCopy needs to send one more request per object or file. (default true)

## Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it's omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)

# azcopy doc

8/22/2022 • 2 minutes to read • [Edit Online](#)

Generates documentation for the tool in Markdown format.

## Synopsis

Generates documentation for the tool in Markdown format, and stores them in the designated location.

By default, the files are stored in a folder named 'doc' inside the current directory.

```
azcopy doc [flags]
```

## Options

`-h` , `--help` help for doc `--output-location` (string) where to put the generated markdown files (default "./doc")

## Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it's omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text").

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;\*.core.cloudapi.

## See also

- [azcopy](#)

# azcopy env

8/22/2022 • 2 minutes to read • [Edit Online](#)

Shows the environment variables that can configure AzCopy's behavior. For a complete list of environment variables, see [AzCopy v10 configuration settings \(Azure Storage\)](#).

## Synopsis

Shows the environment variables that you can use to configure the behavior of AzCopy.

If you set an environment variable by using the command line, that variable will be readable in your command line history. Consider clearing variables that contain credentials from your command line history. To keep variables from appearing in your history, you can use a script to prompt the user for their credentials, and to set the environment variable.

```
azcopy env [flags]
```

## Options

`-h`, `--help` help for env `--show-sensitive` Shows sensitive/secret environment variables.

## Options inherited from parent commands

`--cap-mbps float` Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it's omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;\*.core.cloudapi.de;.core.usgovcloudapi.net;storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)

# azcopy jobs

8/22/2022 • 2 minutes to read • [Edit Online](#)

Subcommands related to managing jobs.

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Examples

```
azcopy jobs show [jobID]
```

## Options

`-h` , `--help` Help for jobs

## Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it is omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;.core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)
- [azcopy jobs list](#)
- [azcopy jobs resume](#)
- [azcopy jobs show](#)

# azcopy jobs clean

8/22/2022 • 2 minutes to read • [Edit Online](#)

Remove all log and plan files for all jobs

```
azcopy jobs clean [flags]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Examples

```
azcopy jobs clean --with-status=completed
```

## Options

`-h` , `--help` help for clean `--with-status` (string) only remove the jobs with this status, available values: All, Canceled, Failed, Completed CompletedWithErrors, CompletedWithSkipped, CompletedWithErrorsAndSkipped (default "All")

## Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it is omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;.core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy jobs](#)

# azcopy jobs list

8/22/2022 • 2 minutes to read • [Edit Online](#)

Displays information on all jobs.

## Synopsis

```
azcopy jobs list [flags]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Options

`-h` , `--help` help for list `--with-status` (string) List the jobs with given status, available values: All, Canceled, Failed, InProgress, Completed, CompletedWithErrors, CompletedWithFailures, CompletedWithErrorsAndSkipped (default "All")

## Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it is omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;.core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy jobs](#)

# azcopy jobs remove

8/22/2022 • 2 minutes to read • [Edit Online](#)

Remove all files associated with the given job ID.

## NOTE

You can customize the location where log and plan files are saved. See the [azcopy env](#) command to learn more.

```
azcopy jobs remove [jobID] [flags]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Examples

```
azcopy jobs rm e52247de-0323-b14d-4cc8-76e0be2e2d44
```

## Options

`--help` Help for remove.

## Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it's omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is

'.core.windows.net;core.chinacloudapi.cn;.core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy jobs](#)

# azcopy jobs resume

8/22/2022 • 2 minutes to read • [Edit Online](#)

Resumes the existing job with the given job ID.

## Synopsis

```
azcopy jobs resume [jobID] [flags]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Options

`--destination-sas` (string) destination SAS token of the destination for a given Job ID.

`--exclude` (string) Filter: exclude these failed transfer(s) when resuming the job. Files should be separated by ':'.

`-h` , `--help` help for resume

`--include` (string) Filter: only include these failed transfer(s) when resuming the job. Files should be separated by ':'.

`--source-sas` (string) Source SAS token of the source for a given Job ID.

## Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it is omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is

'.core.windows.net;core.chinacloudapi.cn;.core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy jobs](#)

# azcopy jobs show

8/22/2022 • 2 minutes to read • [Edit Online](#)

Shows detailed information for the given job ID.

## Synopsis

If only the job ID is supplied without a flag, then the progress summary of the job is returned.

The byte counts and percent complete that appears when you run this command reflect only files that are completed in the job. They don't reflect partially completed files.

If the `--with-status` flag is set, then the list of transfers in the job with the given value will be shown.

```
azcopy jobs show [jobID] [flags]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Options

`-h` , `--help` Help for show `--with-status` (string) Only list the transfers of job with this status, available values: Started, Success, Failed.

## Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it is omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;.core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy jobs](#)

# azcopy list

8/22/2022 • 2 minutes to read • [Edit Online](#)

Lists the entities in a given resource.

## Synopsis

Only Blob containers are supported in the current release.

```
azcopy list [containerURL] [flags]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Examples

```
azcopy list [containerURL] --properties [semicolon(;) separated list of attributes (LastModifiedTime, VersionId, BlobType, BlobAccessTier, ContentType, ContentEncoding, LeaseState, LeaseDuration, LeaseStatus) enclosed in double quotes ("")]
```

## Options

`-h` , `--help` Help for list

`--machine-readable` Lists file sizes in bytes.

`--mega-units` Displays units in orders of 1000, not 1024.

`--properties` (string) delimiter (:) separated values of properties required in list output.

`--running-tally` Counts the total number of files and their sizes.

## Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it's omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;.core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)

# azcopy login

8/22/2022 • 3 minutes to read • [Edit Online](#)

Logs in to Azure Active Directory to access Azure Storage resources.

## Synopsis

Log in to Azure Active Directory to access Azure Storage resources.

To be authorized to your Azure Storage account, you must assign the **Storage Blob Data Contributor** role to your user account in the context of either the Storage account, parent resource group, or parent subscription.

This command will cache encrypted login information for current user using the OS built-in mechanisms.

### IMPORTANT

If you set an environment variable by using the command line, that variable will be readable in your command line history. Consider clearing variables that contain credentials from your command line history. To keep variables from appearing in your history, you can use a script to prompt the user for their credentials, and to set the environment variable.

```
azcopy login [flags]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Tutorial: Migrate on-premises data to cloud storage with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Examples

Log in interactively with default AAD tenant ID set to common:

```
azcopy login
```

Log in interactively with a specified tenant ID:

```
azcopy login --tenant-id "[TenantID]"
```

Log in by using the system-assigned identity of a Virtual Machine (VM):

```
azcopy login --identity
```

Log in by using the user-assigned identity of a VM and a Client ID of the service identity:

```
azcopy login --identity --identity-client-id "[ServiceIdentityClientID]"
```

Log in by using the user-assigned identity of a VM and an Object ID of the service identity:

```
azcopy login --identity --identity-object-id "[ServiceIdentityObjectID]"
```

Log in by using the user-assigned identity of a VM and a Resource ID of the service identity:

```
azcopy login --identity --identity-resource-id
"/subscriptions/<subscriptionId>/resourcegroups/myRG/providers/Microsoft.ManagedIdentity/userAssignedIdentities/myID"
```

Log in as a service principal by using a client secret:

Set the environment variable AZCOPY\_SPA\_CLIENT\_SECRET to the client secret for secret based service principal auth.

```
azcopy login --service-principal --application-id <your service principal's application ID>
```

Log in as a service principal by using a certificate and it's password:

Set the environment variable AZCOPY\_SPA\_CERT\_PASSWORD to the certificate's password for cert based service principal auth

```
azcopy login --service-principal --certificate-path /path/to/my/cert --application-id <your service principal's application ID>
```

Treat /path/to/my/cert as a path to a PEM or PKCS12 file--. AzCopy doesn't reach into the system cert store to obtain your certificate. --certificate-path is mandatory when doing cert-based service principal auth.

Subcommand for login to check the login status of your current session.

```
azcopy login status
```

## Options

--aad-endpoint (string) The Azure Active Directory endpoint to use. The default (<https://login.microsoftonline.com>) is correct for the global Azure cloud. Set this parameter when authenticating in a national cloud. Not needed for Managed Service Identity

--application-id (string) Application ID of user-assigned identity. Required for service principal auth.

--certificate-path (string) Path to certificate for SPN authentication. Required for certificate-based service principal auth.

-h , --help Help for login

--identity Log in using virtual machine's identity, also known as managed service identity (MSI).

--identity-client-id (string) Client ID of user-assigned identity.

--identity-object-id (string) Object ID of user-assigned identity.

--identity-resource-id (string) Resource ID of user-assigned identity.

--service-principal Log in via Service Principal Name (SPN) by using a certificate or a secret. The client secret or certificate password must be placed in the appropriate environment variable. Type AzCopy env to see names and descriptions of environment variables.

--tenant-id (string) The Azure Active Directory tenant ID to use for OAuth device interactive login.

## Options inherited from parent commands

--cap-mbps (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it's omitted, the throughput isn't capped.

--output-type (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

--trusted-microsoft-suffixes (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)

# azcopy login status

8/22/2022 • 2 minutes to read • [Edit Online](#)

Lists the entities in a given resource.

## Synopsis

Prints if you're currently logged in to your Azure Storage account.

```
azcopy login status [flags]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

### Options

`--endpoint` Prints the Azure Active Directory endpoint that is being used in the current session.

`-h` , `--help` Help for status

`--tenant` Prints the Azure Active Directory tenant ID that is currently being used in session.

### Options inherited from parent commands

`--aad-endpoint` (string) The Azure Active Directory endpoint to use. The default (<https://login.microsoftonline.com>) is correct for the global Azure cloud. Set this parameter when authenticating in a national cloud. Not needed for Managed Service Identity

`--application-id` (string) Application ID of user-assigned identity. Required for service principal auth.

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it is omitted, the throughput isn't capped.

`--certificate-path` (string) Path to certificate for SPN authentication. Required for certificate-based service principal auth.

`--identity` Log in using virtual machine's identity, also known as managed service identity (MSI).

`--identity-client-id` (string) Client ID of user-assigned identity.

`--identity-object-id` (string) Object ID of user-assigned identity.

`--identity-resource-id` (string) Resource ID of user-assigned identity.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--service-principal` Log in via Service Principal Name (SPN) by using a certificate or a secret. The client secret or certificate password must be placed in the appropriate environment variable. Type AzCopy env to see names and descriptions of environment variables.

`--tenant-id` (string) The Azure Active Directory tenant ID to use for OAuth device interactive login.

--trusted-microsoft-suffixes (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)

# azcopy logout

8/22/2022 • 2 minutes to read • [Edit Online](#)

Logs the user out and terminates access to Azure Storage resources.

## Synopsis

This command will remove all the cached login information for the current user.

```
azcopy logout [flags]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Options

`-h` , `--help` help for logout

### Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it is omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is

'`.core.windows.net;core.chinacloudapi.cn;core.cloudapi.de;core.usgovcloudapi.net;*.storage.azure.net`'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)

# azcopy make

8/22/2022 • 2 minutes to read • [Edit Online](#)

Creates a container or file share.

## Synopsis

Create a container or file share represented by the given resource URL.

```
azcopy make [resourceURL] [flags]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Examples

```
azcopy make "https://[account-name].[blob,file,dfs].core.windows.net/[top-level-resource-name]"
```

## Options

`-h` , `--help` help for make `--quota-gb` (uint32) Specifies the maximum size of the share in gigabytes (GiB), 0 means you accept the file service's default quota.

## Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it is omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;.core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)

# azcopy remove

8/22/2022 • 3 minutes to read • [Edit Online](#)

Delete blobs or files from an Azure storage account.

## Synopsis

```
azcopy remove [resourceURL] [flags]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Examples

Remove a single blob by using a SAS token:

```
azcopy rm "https://[account].blob.core.windows.net/[container]/[path/to/blob]?[SAS]"
```

Remove an entire virtual directory by using a SAS token:

```
azcopy rm "https://[account].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" --recursive=true
```

Remove only the blobs inside of a virtual directory, but don't remove any subdirectories or blobs within those subdirectories:

```
azcopy rm "https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]" --recursive=false
```

Remove a subset of blobs in a virtual directory (For example: remove only jpg and pdf files, or if the blob name is "exactName"):

```
azcopy rm "https://[account].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" --recursive=true --include-pattern="*.jpg;*.pdf;exactName"
```

Remove an entire virtual directory but exclude certain blobs from the scope (For example: every blob that starts with foo or ends with bar):

```
azcopy rm "https://[account].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" --recursive=true --exclude-pattern="foo*;*bar"
```

Remove specified version IDs of a blob from Azure Storage. Ensure that source is a valid blob and `versionidsfile` which takes in a path to the file where each version is written on a separate line. All the specified versions will be removed from Azure Storage.

```
azcopy rm "https://[srcaccount].blob.core.windows.net/[containername]/[blobname]" "/path/to/dir" --list-of-versions="/path/to/dir/[versionidsfile]"
```

Remove specific blobs and virtual directories by putting their relative paths (NOT URL-encoded) in a file:

```
azcopy rm "https://[account].blob.core.windows.net/[container]/[path/to/parent/dir]" --recursive=true --list-of-files=/usr/bar/list.txt
```

Remove a single file from a Blob Storage account that has a hierarchical namespace (include/exclude not

supported):

```
azcopy rm "https://[account].dfs.core.windows.net/[container]/[path/to/file]?[SAS]"
```

Remove a single directory from a Blob Storage account that has a hierarchical namespace (include/exclude not supported):

```
azcopy rm "https://[account].dfs.core.windows.net/[container]/[path/to/directory]?[SAS]"
```

## Options

**--delete-snapshots** (string) By default, the delete operation fails if a blob has snapshots. Specify 'include' to remove the root blob and all its snapshots; alternatively specify 'only' to remove only the snapshots but keep the root blob.

**--dry-run** Prints the path files that would be removed by the command. This flag doesn't trigger the removal of the files.

**--exclude-path** (string) Exclude these paths when removing. This option doesn't support wildcard characters (\*). Checks relative path prefix. For example: myFolder;myFolder/subDirName/file.pdf

**--exclude-pattern** (string) Exclude files where the name matches the pattern list. For example:  
.jpg;.pdf;exactName

**--force-if-read-only** When deleting an Azure Files file or folder, force the deletion to work even if the existing object has its read-only attribute set

**--from-to** (string) Optionally specifies the source destination combination. For Example: BlobTrash, FileTrash, BlobFSTrash

**-h** , **--help** help for remove

**--include-path** (string) Include only these paths when removing. This option doesn't support wildcard characters (\*). Checks relative path prefix. For example: myFolder;myFolder/subDirName/file.pdf

**--include-pattern** (string) Include only files where the name matches the pattern list. For example:  
.jpg;.pdf;exactName

**--list-of-files** (string) Defines the location of a file which contains the list of files and directories to be deleted. The relative paths should be delimited by line breaks, and the paths should NOT be URL-encoded.

**--list-of-versions** (string) Specifies a file where each version ID is listed on a separate line. Ensure that the source must point to a single blob and all the version IDs specified in the file using this flag must belong to the source blob only. Specified version IDs of the given blob will get deleted from Azure Storage.

**--log-level** (string) Define the log verbosity for the log file. Available levels include: INFO(all requests/responses), WARNING(slow responses), ERROR(only failed requests), and NONE(no output logs). (default 'INFO') (default "INFO")

**--permanent-delete** (string) This is a preview feature that PERMANENTLY deletes soft-deleted snapshots/versions. Possible values include 'snapshots', 'versions', 'snapshotsandversions', 'none'. (default "none")

**--recursive** Look into subdirectories recursively when syncing between directories.

## Options inherited from parent commands

**--cap-mbps float** Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it's omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;.core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)

# azcopy sync

8/22/2022 • 6 minutes to read • [Edit Online](#)

Replicates the source location to the destination location. This article provides a detailed reference for the azcopy sync command. To learn more about synchronizing blobs between source and destination locations, see [Synchronize with Azure Blob storage by using AzCopy v10](#). For Azure Files, see [Synchronize files](#).

## Synopsis

The last modified times are used for comparison. The file is skipped if the last modified time in the destination is more recent. The supported pairs are:

- Local <-> Azure Blob / Azure File (either SAS or OAuth authentication can be used)
- Azure Blob <-> Azure Blob (Source must include a SAS or is publicly accessible; either SAS or OAuth authentication can be used for destination)
- Azure File <-> Azure File (Source must include a SAS or is publicly accessible; SAS authentication should be used for destination)
- Azure Blob <-> Azure File

The sync command differs from the copy command in several ways:

1. By default, the recursive flag is true and sync copies all subdirectories. Sync only copies the top-level files inside a directory if the recursive flag is false.
2. When syncing between virtual directories, add a trailing slash to the path (refer to examples) if there's a blob with the same name as one of the virtual directories.
3. If the 'delete-destination' flag is set to true or prompt, then sync will delete files and blobs at the destination that aren't present at the source.

Advanced:

Note that if you don't specify a file extension, AzCopy automatically detects the content type of the files when uploading from the local disk, based on the file extension or content.

The built-in lookup table is small but on Unix it's augmented by the local system's mime.types file(s) if available under one or more of these names:

- /etc/mime.types
- /etc/apache2/mime.types
- /etc/apache/mime.types

On Windows, MIME types are extracted from the registry.

Also note that sync works off of the last modified times exclusively. So in the case of Azure File <-> Azure File, the header field Last-Modified is used instead of x-ms-file-change-time, which means that metadata changes at the source can also trigger a full copy.

```
azcopy sync [flags]
```

## Examples

Sync a single file:

```
azcopy sync "/path/to/file.txt" "https://[account].blob.core.windows.net/[container]/[path/to/blob]"
```

Same as above, but also compute an MD5 hash of the file content, and then save that MD5 hash as the blob's Content-MD5 property.

```
azcopy sync "/path/to/file.txt" "https://[account].blob.core.windows.net/[container]/[path/to/blob]" --put-md5
```

Sync an entire directory including its subdirectories (note that recursive is by default on):

```
azcopy sync "/path/to/dir" "https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]" or
azcopy sync "/path/to/dir" "https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]" --put-md5
```

Sync only the files inside of a directory but not subdirectories or the files inside of subdirectories:

```
azcopy sync "/path/to/dir" "https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]" --recursive=false
```

Sync a subset of files in a directory (For example: only jpg and pdf files, or if the file name is "exactName"):

```
azcopy sync "/path/to/dir" "https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]" --include-pattern="*.jpg;*.pdf;exactName"
```

Sync an entire directory but exclude certain files from the scope (For example: every file that starts with foo or ends with bar):

```
azcopy sync "/path/to/dir" "https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]" --exclude-pattern="foo*;*bar"
```

Sync a single blob:

```
azcopy sync "https://[account].blob.core.windows.net/[container]/[path/to/blob]?[SAS]"
"https://[account].blob.core.windows.net/[container]/[path/to/blob]"
```

Sync a virtual directory:

```
azcopy sync "https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]?[SAS]"
"https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]" --recursive=true
```

Sync a virtual directory that has the same name as a blob (add a trailing slash to the path in order to disambiguate):

```
azcopy sync "https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]/?[SAS]"
"https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]/" --recursive=true
```

Sync an Azure File directory (same syntax as Blob):

```
azcopy sync "https://[account].file.core.windows.net/[share]/[path/to/dir]?[SAS]"
"https://[account].file.core.windows.net/[share]/[path/to/dir]" --recursive=true
```

Note: if include and exclude flags are used together, only files matching the include patterns are used, but those matching the exclude patterns are ignored.

## Options

`--block-size-mb` (float) Use this block size (specified in MiB) when uploading to Azure Storage or downloading from Azure Storage. Default is automatically calculated based on file size. Decimal fractions are allowed (For example: 0.25).

`--check-md5` (string) Specifies how strictly MD5 hashes should be validated when downloading. This option is only available when downloading. Available values include: NoCheck, LogOnly, FailIfDifferent, FailIfDifferentOrMissing. (default 'FailIfDifferent'). (default "FailIfDifferent")

`--cpk-by-name` (string) Client provided key by name let clients that make requests against Azure Blob storage an

option to provide an encryption key on a per-request basis. Provided key name will be fetched from Azure Key Vault and will be used to encrypt the data

**--cpk-by-value** Client provided key by name let clients that make requests against Azure Blob storage an option to provide an encryption key on a per-request basis. Provided key and its hash will be fetched from environment variables

**--delete-destination** (string) Defines whether to delete extra files from the destination that aren't present at the source. Could be set to true, false, or prompt. If set to prompt, the user will be asked a question before scheduling files and blobs for deletion. (default 'false'). (default "false")

**--dry-run** Prints the path of files that would be copied or removed by the sync command. This flag doesn't copy or remove the actual files.

**--exclude-attributes** (string) (Windows only) Exclude files whose attributes match the attribute list. For example: A;S;R

**--exclude-path** (string) Exclude these paths when comparing the source against the destination. This option doesn't support wildcard characters (\*). Checks relative path prefix(For example: myFolder;myFolder/subDirName/file.pdf).

**--exclude-pattern** (string) Exclude files where the name matches the pattern list. For example: jpg;pdf;exactName

**--exclude-regex** (string) Exclude the relative path of the files that match with the regular expressions. Separate regular expressions with ':'.

**--from-to** (string) Optionally specifies the source destination combination. For Example: LocalBlob, BlobLocal, LocalFile, FileLocal, BlobFile, FileBlob, etc.

**-h , --help** help for sync

**--include-attributes** (string) (Windows only) Include only files whose attributes match the attribute list. For example: A;S;R

**--include-pattern** (string) Include only files where the name matches the pattern list. For example: jpg;pdf;exactName

**--include-regex** (string) Include the relative path of the files that match with the regular expressions. Separate regular expressions with ':'.

**--log-level** (string) Define the log verbosity for the log file, available levels: INFO(all requests and responses), WARNING(slow responses), ERROR(only failed requests), and NONE(no output logs). (default INFO). (default "INFO")

**--mirror-mode** Disable last-modified-time based comparison and overwrites the conflicting files and blobs at the destination if this flag is set to true. Default is false

**--preserve-permissions** False by default. Preserves ACLs between aware resources (Windows and Azure Files, or ADLS Gen 2 to ADLS Gen 2). For Hierarchical Namespace accounts, you'll need a container SAS or OAuth token with Modify Ownership and Modify Permissions permissions. For downloads, you'll also need the

**--backup** flag to restore permissions where the new Owner won't be the user running AzCopy. This flag applies to both files and folders, unless a file-only filter is specified (for example, include-pattern).

**--preserve-smb-info** For SMB-aware locations, flag will be set to true by default. Preserves SMB property info (last write time, creation time, attribute bits) between SMB-aware resources (Azure Files). This flag applies to both files and folders, unless a file-only filter is specified (for example, include-pattern). The info transferred for folders is the same as that for files, except for Last Write Time that isn't preserved for folders. (default true)

--put-md5 Create an MD5 hash of each file, and save the hash as the Content-MD5 property of the destination blob or file. (By default the hash is NOT created.) Only available when uploading.

--recursive True by default, look into subdirectories recursively when syncing between directories. (default true). (default true)

--s2s-preserve-access-tier Preserve access tier during service to service copy. Refer to [Azure Blob storage: hot, cool, and archive access tiers](#) to ensure destination storage account supports setting access tier. In the cases that setting access tier isn't supported, please use s2sPreserveAccessTier=false to bypass copying access tier. (default true). (default true)

--s2s-preserve-blob-tags Preserve index tags during service to service sync from one blob storage to another

## Options inherited from parent commands

--cap-mbps (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it's omitted, the throughput isn't capped.

--output-type (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

--trusted-microsoft-suffixes (string) Specifies other domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;.core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)

# azcopy set-properties (preview)

8/22/2022 • 3 minutes to read • [Edit Online](#)

Given a location, change all the valid system properties of that storage (blob or file).

## Synopsis

```
azcopy set-properties [resourceURL] [flags]
```

Sets properties of Blob and File storage. The properties currently supported by this command are:

- Blobs -> Tier, Metadata, Tags
- Data Lake Storage Gen2 -> Tier, Metadata, Tags
- Files -> Metadata

### NOTE

Data Lake Storage Gen2 endpoints will be replaced by Blob Storage endpoints.

Refer to the examples for more information.

## Related conceptual articles

- [Get started with AzCopy](#)
- [Replace blob properties and metadata by using AzCopy v10 \(preview\)](#)

## Examples

Change tier of blob to hot:

```
azcopy set-properties "https://[account].blob.core.windows.net/[container]/[path/to/blob]" --block-blob-tier=hot
```

Change tier of blob from archive to cool with rehydrate priority set to high:

```
azcopy set-properties "https://[account].blob.core.windows.net/[container]/[path/to/blob]" --block-blob-tier=cool --rehydrate-priority=high
```

Change tier of all files in a directory to archive:

```
azcopy set-properties "https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]" --block-blob-tier=archive --recursive=true
```

Change metadata of blob to {key = "abc", val = "def"} and {key = "ghi", val = "jkl"}:

```
azcopy set-properties "https://[account].blob.core.windows.net/[container]/[path/to/blob]" --metadata=abc=def;ghi=jkl
```

Change metadata of all files in a directory to {key = "abc", val = "def"} and {key = "ghi", val = "jkl"}:

```
azcopy set-properties "https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]" --metadata=abc=def;ghi=jkl --recursive=true
```

Clear all existing metadata of blob:

```
azcopy set-properties "https://[account].blob.core.windows.net/[container]/[path/to/blob]" --metadata=clear
```

Change blob-tags of blob to {key = "abc", val = "def"} and {key = "ghi", val = "jkl"}:

```
azcopy set-properties "https://[account].blob.core.windows.net/[container]/[path/to/blob]" --blob-tags=abc=def&ghi=jkl
```

While setting tags on the blobs, there are other permissions('t' for tags) with SAS. Without those tags, the service will return an authorization error.

Clear all existing blob-tags of blob:

```
azcopy set-properties "https://[account].blob.core.windows.net/[container]/[path/to/blob]" --blob-tags=clear
```

While setting tags on the blobs, there are other permissions('t' for tags) with SAS. Without those tags, the service will return an authorization error.

## Options

- blob-tags string Set tags on blobs to categorize data in your storage account (separated by '&')
- block-blob-tier string Changes the access tier of the blobs to the given tier (default "None")
- dry-run Prints the file paths that would be affected by this command. This flag doesn't affect the actual files.
- exclude-path string Exclude these paths when removing. This option doesn't support wildcard characters (\*). Checks relative path prefix. For example: myFolder;myFolder/subDirName/file.pdf
- exclude-pattern string Exclude files where the name matches the pattern list. For example:  
.jpg;.pdf;exactName
- from-to string Optionally specifies the source destination combination. Valid values: BlobNone, FileNone, BlobFSNone
- h , --help help for set-properties
- include-path string Include only these paths when setting property. This option doesn't support wildcard characters (\*). Checks relative path prefix. For example: myFolder;myFolder/subDirName/file.pdf
- include-pattern string Include only files where the name matches the pattern list. For example:  
.jpg;.pdf;exactName
- list-of-files string Defines the location of the text file that has the list of files to be copied.
- metadata string Set the given location with these key-value pairs (separated by ';') as metadata.
- page-blob-tier string Upload page blob to Azure Storage using this blob tier. (default 'None'). (default "None")
- recursive Look into subdirectories recursively when uploading from local file system.
- rehydrate-priority string Optional flag that sets rehydrate priority for rehydration. Valid values: Standard, High. Default- standard (default "Standard")

## Options inherited from parent commands

- cap-mbps float Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it's omitted, the throughput isn't capped.
- log-level (string) Define the log verbosity for the log file, available levels: INFO(all requests/responses), WARNING(slow responses), ERROR(only failed requests), and NONE(no output logs). (default 'INFO'). (default

"INFO")

--output-type (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

--output-level (string) Define the output verbosity. Available levels: essential, quiet. (default "default")

--trusted-microsoft-suffixes (string) Specifies other domain suffixes where Azure Active Directory log in tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)

# How to configure Blobfuse2 (preview)

8/22/2022 • 2 minutes to read • [Edit Online](#)

## IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

If you need to use BlobFuse in a production environment, BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 setup documentation](#)
- [The BlobFuse v1 project on GitHub](#)

BlobFuse2 uses a variety of configuration settings to control its behaviors, including:

- Access to a storage blob
- Logging
- Pipeline engagement
- Caching behavior
- Permissions

For a complete list of settings and their descriptions, see [the base configuration file on GitHub](#).

There are 3 ways of managing configuration settings for BlobFuse2 (in order of precedence):

1. [CLI parameters](#)
2. [Environment variables](#)
3. [A configuration file](#)

Using a configuration file is the preferred method, but the other methods can be useful in some circumstances.

## Configuration file

Creating a configuration file is the preferred method of establishing settings for BlobFuse2. Once you have provided the desired settings in the file, reference the configuration file when using the `blobfuse2 mount` or other commands. Example:

```
blobfuse2 mount ./mount --config-file=~/config.yaml
```

The [Base BlobFuse2 configuration file](#) contains a complete list of all settings and a brief explanation of each.

Use the [Sample file cache configuration file](#) or the [sample streaming configuration file](#) to get started quickly using some basic settings for each of those scenarios.

## Environment variables

Setting environment variables is another way to configure some BlobFuse2 settings. The supported environment variables are useful for specifying the blob storage container to be accessed and the authorization method.

For more details on using environment variables, see [The environment variables documentation](#)

See [the BlobFuse2 README](#) for a complete list of variables that can be used.

## CLI Parameters

Configuration settings can be set when passed as parameters of the BlobFuse2 command set, such as the `blobfuse2 mount` command. The mount command typically references a configuration file that contains all of the settings, but individual settings in the configuration file can be overridden by CLI parameters. In this example, the config.yaml configuration file is referenced, but the container to be mounted and the logging options are overridden:

```
blobfuse2 mount ./mount_dir --config-file=./config.yaml --container-name=blobfuse2b --log-level=log_debug --log-file-path=./blobfuse2b.log
```

For more information about the complete BlobFuse2 command set, including the `blobfuse2 mount` command, see [The BlobFuse2 command set reference \(preview\)](#) and [The BlobFuse2 mount command reference \(preview\)](#).

## See also

- [What is BlobFuse2? \(preview\)](#)
- [How to mount an Azure blob storage container on Linux with BlobFuse2 \(preview\)](#)

# How to use the BlobFuse2 command set (preview)

8/22/2022 • 2 minutes to read • [Edit Online](#)

This reference shows how to use the BlobFuse2 command set to mount Azure blob storage containers as file systems on Linux, and how to manage them.

## IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

If you need to use BlobFuse in a production environment, BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 setup documentation](#)
- [The BlobFuse v1 project on GitHub](#)

## Syntax

The `blobfuse2` command has 2 formats:

```
blobfuse2 --[flag-name]=[flag-value]
```

```
blobfuse2 [command] [arguments] --[flag-name]=[flag-value]
```

## Flags (Options)

Most flags are specific to individual BlobFuse2 commands. See the documentation for [each command](#) for details and examples.

The following options can be used without a command or are inherited by individual commands:

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
disable-version-check		boolean	false	Whether to disable the automatic BlobFuse2 version check
help	-h	n/a	n/a	Help info for the blobfuse2 command and subcommands
version	-v	n/a	n/a	Display BlobFuse2 version information

## Commands

The supported commands for BlobFuse2 are:

COMMAND	DESCRIPTION
<a href="#">mount</a>	Mounts an Azure blob storage container as a filesystem in Linux or lists mounted file systems
<a href="#">mountv1</a>	Mounts a blob container using legacy BlobFuse configuration and CLI parameters
<a href="#">unmount</a>	Unmounts a BlobFuse2-mounted file system
<a href="#">completion</a>	Generates an autocompletion script for BlobFuse2 for the specified shell
<a href="#">secure</a>	Encrypts or decrypts a configuration file, or gets or sets values in an encrypted configuration file
<a href="#">version</a>	Displays the current version of BlobFuse2
<a href="#">help</a>	Gives help information about any command

## Arguments

BlobFuse2 command arguments are specific to the individual commands. See the documentation for [each command](#) for details and examples.

## See also

- [What is BlobFuse2? \(preview\)](#)
- [How to mount an Azure blob storage container on Linux with BlobFuse2 \(preview\)](#)
- [BlobFuse2 configuration reference \(preview\)](#)
- [How to troubleshoot BlobFuse2 issues \(preview\)](#)

# How to use the BlobFuse2 mount command (preview)

8/22/2022 • 3 minutes to read • [Edit Online](#)

Use the `blobfuse2 mount` command to mount a blob storage container as a file system in Linux, or to display existing mount points.

## IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

If you need to use BlobFuse in a production environment, BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 setup documentation](#)
- [The BlobFuse v1 project on GitHub](#)

## Command Syntax

The `blobfuse2 mount` command has 2 formats:

```
blobfuse2 mount [path] --[flag-name]=[flag-value]
```

```
blobfuse2 mount [command] --[flag-name]=[flag-value]
```

## Arguments

[path]

Specify a file path to the directory where the storage container will be mounted. Example:

```
blobfuse2 mount ./mount_path ...
```

[command]

The supported subcommands for `blobfuse2 mount` are:

COMMAND	DESCRIPTION
<a href="#">all</a>	Mounts all azure blob containers in a specified storage account
<a href="#">list</a>	Lists all BlobFuse2 mount points

Select one of the command links in the table above to view the documentation for the individual subcommands, including the arguments and flags they support.

## Flags (options)

Some flags are inherited from the parent command, `blobfuse2`, and others only apply to the `blobfuse2 mount` command.

### Flags inherited from the BlobFuse2 command

The following flags are inherited from parent command `blobfuse2`):

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
disable-version-check		boolean	false	Enables or disables automatic version checking of the BlobFuse2 binaries
help	-h	n/a	n/a	Help info for the blobfuse2 command and subcommands

### Flags that apply only to the BlobFuse2 mount command

The following flags apply only to command `blobfuse2 mount`:

FLAG	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
allow-other	boolean	false	Allow other users to access this mount point
attr-cache-timeout	uint32	120	Attribute cache timeout (in seconds)
attr-timeout	uint32		Attribute timeout (in seconds)
config-file	string	./config.yaml	The path for the file where the account credentials are provided Default is config.yaml in current directory.
container-name	string		The name of the container to be mounted
entry-timeout	uint32		Entry timeout (in seconds)
file-cache-timeout	uint32	120	File cache timeout (in seconds)
foreground	boolean	false	Whether the file system is mounted in foreground mode
log-file-path	string	\$HOME/.blobfuse2/blobfuse2.log	The path for log files

FLAG	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
log-level	LOG_OFF LOG_CRIT LOG_ERR LOG_WARNING LOG_INFO LOG_DEBUG LOG_WARNING	LOG_WARNING	The level of logging written to <code>--log-file-path</code> .
negative-timeout	uint32		The negative entry timeout (in seconds)
no-symlinks	boolean	false	Whether or not symlinks should be supported
passphrase	string		Key to decrypt config file. Can also be specified by env-variable <code>BLOBFUSE2_SECURE_CONFIG_PASSPHRASE</code> . The key length shall be 16 (AES-128), 24 (AES-192), or 32 (AES-256) bytes in length.
read-only	boolean	false	Mount the system in read only mode
secure-config	boolean	false	Encrypt auto generated config file for each container
tmp-path	string	n/a	Configures the tmp location for the cache. (Configure the fastest disk (SSD or ramdisk) for best performance).

## Examples

### NOTE

The following examples assume you have already created a configuration file in the current directory.

1. Mount an individual Azure blob storage container to a new directory using the settings from a configuration file, and with foreground mode disabled:

```
~$ mkdir bf2a
~$ blobfuse2 mount ./bf2a --config-file=../config.yaml --foreground=false

~$ blobfuse2 mount list
1 : /home/<user>/bf2a
```

2. Mount all blob storage containers in the storage account specified in the configuration file to the path specified in the command. (Each container will be a subdirectory under the directory specified):

```
~$ mkdir bf2all
~$ blobfuse2 mount all ./bf2all --config-file=./config.yaml
Mounting container : blobfuse2a to path : bf2all/blobfuse2a
Mounting container : blobfuse2b to path : bf2all/blobfuse2b

~$ blobfuse2 mount list
1 : /home/<user>/bf2all/blobfuse2a
2 : /home/<user>/bf2all/blobfuse2b
```

3. Mount a fast storage device, then mount a blob storage container specifying the path to the mounted disk as the BlobFuse2 file caching location:

```
~$ sudo mkdir /mnt/resource/blobfuse2tmp -p
~$ sudo chown <youruser> /mnt/resource/blobfuse2tmp
~$ mkdir bf2a
~$ blobfuse2 mount ./bf2a --config-file=./config.yaml --tmp-path=/mnt/resource/blobfuse2tmp

~$ blobfuse2 mount list
1 : /home/<user>/bf2a/blobfuse2a
```

4. Mount a blob storage container in read-only mode and skipping the automatic BlobFuse2 version check:

```
blobfuse2 mount ./mount_dir --config-file=./config.yaml --read-only --disable-version-check=true
```

5. Mount a blob storage container using an existing configuration file, but override the container name (mounting another container in the same storage account):

```
blobfuse2 mount ./mount_dir2 --config-file=./config.yaml --container-name=container2
```

## See also

- [The Blobfuse2 mount all command \(preview\)](#)
- [The Blobfuse2 mount list command \(preview\)](#)
- [The Blobfuse2 unmount command \(preview\)](#)
- [The Blobfuse2 mountv1 command \(preview\)](#)

# How to use the BlobFuse2 mount all command to mount all blob containers in a storage account as a Linux file system (preview)

8/22/2022 • 2 minutes to read • [Edit Online](#)

Use the `BlobFuse2 mount all` command to mount all blob containers in a storage account as a Linux file system. Each container will be mounted to a unique subdirectory under the path specified. The subdirectory names will correspond to the container names.

## IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

If you need to use BlobFuse in a production environment, BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 setup documentation](#)
- [The BlobFuse v1 project on GitHub](#)

## Syntax

```
blobfuse2 mount all [path] --[flag-name]=[flag-value]
```

## Arguments

[path]

Specify a file path to the directory where all of the blob storage containers in the storage account will be mounted. Example:

```
blobfuse2 mount all ./mount_path ...
```

## Flags (options)

Flags that apply to `blobfuse2 mount all` are inherited from the parent commands, `blobfuse2` and `blobfuse2 mount`.

### Flags inherited from the BlobFuse2 command

The following flags are inherited from grandparent command `blobfuse2`:

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
disable-version-check		boolean	false	Enables or disables automatic version checking of the BlobFuse2 binaries

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
help	-h	n/a		Help info for the blobfuse2 command and subcommands

### Flags inherited from the BlobFuse2 mount command

The following flags are inherited from parent command `blobfuse2 mount` :

FLAG	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
allow-other	boolean	false	Allow other users to access this mount point
attr-cache-timeout	uint32	120	Attribute cache timeout (in seconds)
attr-timeout	uint32		Attribute timeout (in seconds)
config-file	string	./config.yaml	The path for the file where the account credentials are provided Default is config.yaml in current directory.
container-name	string		The name of the container to be mounted
entry-timeout	uint32		Entry timeout (in seconds)
file-cache-timeout	uint32	120	File cache timeout (in seconds)
foreground	boolean	false	Whether the file system is mounted in foreground mode
log-file-path	string	\$HOME/.blobfuse2/blobfuse2.log	The path for log files
log-level	LOG_OFF LOG_CRIT LOG_ERR LOG_WARNING LOG_INFO LOG_DEBUG LOG_WARNING	LOG_WARNING	The level of logging written to <code>--log-file-path</code> .
negative-timeout	uint32		The negative entry timeout (in seconds)
no-symlinks	boolean	false	Whether or not symlinks should be supported

FLAG	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
passphrase	string		Key to decrypt config file. Can also be specified by env-variable BLOBFUSE2_SECURE_CONFIG_PASSPHRASE The key length shall be 16 (AES-128), 24 (AES-192), or 32 (AES-256) bytes in length.
read-only	boolean	false	Mount the system in read only mode
secure-config	boolean	false	Encrypt auto generated config file for each container
tmp-path	string	n/a	Configures the tmp location for the cache. (Configure the fastest disk (SSD or ramdisk) for best performance).

## Examples

### NOTE

The following examples assume you have already created a configuration file in the current directory.

Mount all blob storage containers in the storage account specified in the configuration file to the path specified in the command. (Each container will be a subdirectory under the directory specified):

```
~$ mkdir bf2all
~$ blobfuse2 mount all ./bf2all --config-file=~/config.yaml
Mounting container : blobfuse2a to path : bf2all/blobfuse2a
Mounting container : blobfuse2b to path : bf2all/blobfuse2b

~$ blobfuse2 mount list
1 : /home/<user>/bf2all/blobfuse2a
2 : /home/<user>/bf2all/blobfuse2b
```

## See also

- [The Blobfuse2 unmount all command \(preview\)](#)
- [The Blobfuse2 mount command \(preview\)](#)
- [The Blobfuse2 unmount command \(preview\)](#)

# How to use the BlobFuse2 mount list command to display all BlobFuse2 mount points (preview)

8/22/2022 • 2 minutes to read • [Edit Online](#)

Use the `BlobFuse2 mount list` command to display all existing BlobFuse2 mount points.

## IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

If you need to use BlobFuse in a production environment, BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 setup documentation](#)
- [The BlobFuse v1 project on GitHub](#)

## Syntax

```
blobfuse2 mount list --[flag-name]=[flag-value]
```

## Flags (options)

Flags that apply to `blobfuse2 mount all` are inherited from the parent commands, `blobfuse2` and `blobfuse2 mount`.

### Flags inherited from the BlobFuse2 command

The following flags are inherited from grandparent command `blobfuse2`:

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
disable-version-check		boolean	false	Enables or disables automatic version checking of the BlobFuse2 binaries
help	-h	n/a		Help info for the blobfuse2 command and subcommands

### Flags inherited from the BlobFuse2 mount command

The following flags are inherited from parent command `blobfuse2 mount`:

FLAG	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
allow-other	boolean	false	Allow other users to access this mount point

FLAG	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
attr-cache-timeout	uint32	120	Attribute cache timeout (in seconds)
attr-timeout	uint32		Attribute timeout (in seconds)
config-file	string	./config.yaml	The path for the file where the account credentials are provided Default is config.yaml in current directory.
container-name	string		The name of the container to be mounted
entry-timeout	uint32		Entry timeout (in seconds)
file-cache-timeout	uint32	120	File cache timeout (in seconds)
foreground	boolean	false	Whether the file system is mounted in foreground mode
log-file-path	string	\$HOME/.blobfuse2/blobfuse2.log	The path for log files
log-level	LOG_OFF LOG_CRIT LOG_ERR LOG_WARNING LOG_INFO LOG_DEBUG LOG_WARNING	LOG_WARNING	The level of logging written to <code>--log-file-path</code> .
negative-timeout	uint32		The negative entry timeout (in seconds)
no-symlinks	boolean	false	Whether or not symlinks should be supported
passphrase	string		Key to decrypt config file. Can also be specified by env-variable BLOBFUSE2_SECURE_CONFIG_PASSPHRASE The key length shall be 16 (AES-128), 24 (AES-192), or 32 (AES-256) bytes in length.
read-only	boolean	false	Mount the system in read only mode

FLAG	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
secure-config	boolean	false	Encrypt auto generated config file for each container
tmp-path	string	n/a	Configures the tmp location for the cache. (Configure the fastest disk (SSD or ramdisk) for best performance).

## Examples

Display all current BlobFuse2 mount points:

```
~$ blobfuse2 mount list
1 : /home/<user>/bf2a
2 : /home/<user>/bf2b
```

## See also

- [The `Blobfuse2 unmount` command \(preview\)](#)
- [The `Blobfuse2 mount` command \(preview\)](#)
- [The `Blobfuse2 mount all` command \(preview\)](#)

# How to use the BlobFuse2 mountv1 command (preview)

8/22/2022 • 3 minutes to read • [Edit Online](#)

Use the `blobfuse2 mountv1` command to generate a configuration file for BlobFuse2 from a BlobFuse v1 configuration file.

## IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 project on GitHub](#)
- [The BlobFuse v1 setup documentation](#)

## Syntax

```
blobfuse2 mountv1 [path] --[flag-name]=[flag-value]
```

## Arguments

[path]

Specify a file path to the directory where the storage container will be mounted. Example:

```
blobfuse2 mountv1 ./mount_path ...
```

## Flags (options)

Some flags are inherited from the parent command, `blobfuse2`, and others only apply to the `blobfuse2 mountv1` command.

### Flags inherited from the BlobFuse2 command

The following flags are inherited from parent command `blobfuse2`):

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
disable-version-check		boolean	false	Enables or disables automatic version checking of the BlobFuse2 binaries
help	-h	n/a	n/a	Help info for the blobfuse2 command and subcommands

### Flags that apply only to the BlobFuse2 mountv1 command

The following flags apply only to command `blobfuse2 mountv1` command:

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
background-download		boolean	false	File download to run in the background on open call
basic-remount-check		boolean	false	Check for an already mounted status using /etc/mtab
block-size-mb		uint		Size of a block to be downloaded during streaming (in MB)
ca-cert-file		string		Specifies the proxy pem certificate path if it's not in the default path
cache-on-list		boolean	true	Cache attributes on listing
cache-poll-timeout-msec		uint		Time in milliseconds in order to poll for possible expired files awaiting cache eviction (in milliseconds)
cache-size-mb		float		File cache size (in MB)
cancel-list-on-mount-seconds		uint16		A list call to the container is by default issued on mount (in seconds)
config-file		string	./config.cfg	Input BlobFuse configuration file
container-name		string		Required if no configuration file is specified
convert-config-only		boolean		Don't mount - only convert v1 configuration to v2
d	-d	boolean	false	Mount with foreground and FUSE logs on

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
empty-dir-check		boolean	false	Disallow remounting using a non-empty tmp-path
enable-gen1		boolean	false	To enable Gen1 mount
file-cache-timeout-in-seconds		uint32	120	During this time, blobfuse will not check whether the file is up to date or not (in seconds)
high-disk-threshold		uint32		High disk threshold (as a percentage)
http-proxy		string		HTTP Proxy address
https-proxy		string		HTTPS Proxy address
invalidate-on-sync		boolean	true	Invalidate file/dir on sync/fsync
log-level		LOG_OFF LOG_CRIT LOG_ERR LOG_WARNING LOG_INFO LOG_DEBUG LOG_WARNING	LOG_WARNING	The level of logging written to syslog.
low-disk-threshold		uint32		Low disk threshold (as a percentage)
max-blocks-per-file		int		Maximum number of blocks to be cached in memory for streaming
max-concurrency		uint16		Option to override default number of concurrent storage connections
max-eviction		uint32		Number of files to be evicted from cache at once
max-retry		int32		Maximum retry count if the failure codes are retryable

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
max-retry-interval-in-seconds		int32		Maximum length of time between 2 retries (in seconds)
no-symlinks		boolean	false	Whether or not symlinks should be supported
o	-o	strings		FUSE options
output-file		string	./config.yaml	Output Blobfuse configuration file
pre-mount-validate		boolean	true	Validate blobfuse2 is mounted
required-free-space-mb		int		Required free space (in MB)
retry-delay-factor		int32		Retry delay between two tries (in seconds)
set-content-type		boolean	false	Turns on automatic 'content-type' property based on the file extension
stream-cache-mb		uint		Limit total amount of data being cached in memory to conserve memory footprint of blobfuse (in MB)
streaming		boolean	false	Enable Streaming
tmp-path		string	n/a	Configures the tmp location for the cache. (Configure the fastest disk (SSD or ramdisk) for best performance).
upload-modified-only		boolean	false	Turn off unnecessary uploads to storage
use-adls		boolean	false	Enables blobfuse to access Azure DataLake storage account

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
use-attr-cache		boolean	false	Enable attribute cache
use-https		boolean	false	Enables HTTPS communication with Blob storage

## Examples

1. Mount a blob container in an Azure Data Lake Storage account using a BlobFuse v1 configuration file:

```
blobfuse2 mountv1 ./mount_dir --config-file=./config.cfg --use-adls=true
```

2. Create a BlobFuse2 configuration file from a v1 configuration file in the same directory, but do not mount any containers:

```
blobfuse2 mountv1 --config-file=./config.cfg --output-file=./config.yaml --convert-config-only=true
```

## See also

- [The Blobfuse2 mount command \(preview\)](#)
- [The Blobfuse2 unmount command \(preview\)](#)
- [The Blobfuse2 command set \(preview\)](#)

# How to use the BlobFuse2 unmount command (preview)

8/22/2022 • 2 minutes to read • [Edit Online](#)

Use the `blobfuse2 unmount` command to unmount one or more existing BlobFuse2 mount points.

## IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

If you need to use BlobFuse in a production environment, BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 setup documentation](#)
- [The BlobFuse v1 project on GitHub](#)

## Syntax

The `blobfuse2 unmount` command has 2 formats:

```
blobfuse2 unmount [mount path] [flags]
```

```
blobfuse2 unmount all [flags]
```

## Arguments

```
[mount path]
```

Specify a file path to the directory that contains the mount point to be unmounted. Example:

```
blobfuse2 unmount ./mount_path ...
```

```
all
```

Unmount all existing BlobFuse2 mount points.

## Flags (options)

The following flags were inherited from the parent (the BlobFuse2 command):

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	EXAMPLE	DESCRIPTION
disable-version-check		boolean	false	--disable-version-check=true	Enables or disables automatic version checking of the BlobFuse2 binaries

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	EXAMPLE	DESCRIPTION
help	-h	n/a		-h or --help	Help info for the blobfuse2 command and subcommands

There are no flags only supported by the `umount` command.

## Examples

1. Unmount a BlobFuse2 mount instance:

```
blobfuse2 umount ./mount_path
```

(Alternatively, you can use a native Linux command to do the same):

```
sudo fusermount3 -u ./mount_path
```

2. Unmount all BlobFuse2 mount points (see also [The BlobFuse2 unmount all command](#)):

```
blobfuse2 umount all
```

## See also

- [The Blobfuse2 unmount all command \(preview\)](#)
- [The Blobfuse2 mount command \(preview\)](#)

# How to use the BlobFuse2 unmount all command to unmount all existing mount points (preview)

8/22/2022 • 2 minutes to read • [Edit Online](#)

Use the `blobfuse2 unmount all` command to unmount all existing BlobFuse2 mount points.

## IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

If you need to use BlobFuse in a production environment, BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 setup documentation](#)
- [The BlobFuse v1 project on GitHub](#)

## Syntax

```
blobfuse2 unmount all --[flag-name]=[flag-value]
```

## Flags (options)

Flags that apply to `blobfuse2 unmount all` are inherited from the grandparent command, [blobfuse2](#).

### Flags inherited from BlobFuse2

The following flags are inherited from grandparent command [blobfuse2](#):

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
disable-version-check		boolean	false	Enables or disables automatic version checking of the BlobFuse2 binaries
help	-h	n/a		Help info for the blobfuse2 command and subcommands

## Examples

Unmount all BlobFuse2 mount points:

```
blobfuse2 unmount all
```

## See also

- [The Blobfuse2 unmount command \(preview\)](#)

- The Blobfuse2 mount all command (preview)

# BlobFuse2 completion command (preview)

8/22/2022 • 2 minutes to read • [Edit Online](#)

Use the `blobfuse2 completion` command to generate the autocompletion script for BlobFuse2 for a specified shell.

## IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 project on GitHub](#)
- [The BlobFuse v1 setup documentation](#)

## Syntax

```
blobfuse2 completion [command] --[flag-name]=[flag-value]
```

## Arguments

```
[command]
```

The supported subcommands for `blobfuse2 completion` are:

COMMAND	DESCRIPTION
bash	Generate the autocompletion script for bash
fish	Generate the autocompletion script for fish
powershell	Generate the autocompletion script for PowerShell
zsh	Generate the autocompletion script for zsh

Select one of the command links in the table above to view the documentation for the individual subcommands, including how to use the generated script.

## Flags (options)

Flags that apply to `blobfuse2 completion` are inherited from the parent command, `blobfuse2`, or apply only to the `blobfuse2 completion` subcommands.

### Flags inherited from the BlobFuse2 command

The following flags are inherited from parent command `blobfuse2`:

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
disable-version-check		boolean	false	Enables or disables automatic version checking of the BlobFuse2 binaries
help	-h	n/a		Help info for the blobfuse2 command and subcommands

## See also

- [The Blobfuse2 completion bash command \(preview\)](#)
- [The Blobfuse2 completion fish command \(preview\)](#)
- [The Blobfuse2 completion PowerShell command \(preview\)](#)
- [The Blobfuse2 completion zsh command \(preview\)](#)

# BlobFuse2 completion bash command (preview)

8/22/2022 • 2 minutes to read • [Edit Online](#)

Use the `blobfuse2 completion bash` command to generate the autocompletion script for BlobFuse2 for the bash shell.

## IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 project on GitHub](#)
- [The BlobFuse v1 setup documentation](#)

## Syntax

```
blobfuse2 completion bash --[flag-name]=[flag-value]
```

## Flags (options)

Flags that apply to `blobfuse2 completion bash` are inherited from the grandparent command, `blobfuse2`, or apply only to the `blobfuse2 completion` subcommands.

### Flags inherited from the BlobFuse2 command

The following flags are inherited from grandparent command `blobfuse2`:

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
disable-version-check		boolean	false	Enables or disables automatic version checking of the BlobFuse2 binaries
help	-h	n/a		Help info for the blobfuse2 command and subcommands

### Flags that apply to the BlobFuse2 completion subcommands

The following flags apply only to the `blobfuse2 completion` subcommands:

FLAG	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
no-descriptions	boolean	false	Disable completion descriptions

## Usage

The generated script depends on the 'bash-completion' package. If it is not installed already, you can install it via

your OS's package manager.

To load completions in your current shell session:

```
source <(blobfuse2 completion bash)
```

To load completions for every new session, execute once:

- On Linux:

```
blobfuse2 completion bash > /etc/bash_completion.d/blobfuse2
```

- On macOS:

```
blobfuse2 completion bash > /usr/local/etc/bash_completion.d/blobfuse2
```

**NOTE**

You will need to start a new shell for this setup to take effect.

## See also

- [The Blobfuse2 completion command \(preview\)](#)
- [The Blobfuse2 completion fish command \(preview\)](#)
- [The Blobfuse2 completion PowerShell command \(preview\)](#)
- [The Blobfuse2 completion zsh command \(preview\)](#)

# BlobFuse2 completion fish command (preview)

8/22/2022 • 2 minutes to read • [Edit Online](#)

Use the `blobfuse2 completion fish` command to generate the autocompletion script for BlobFuse2 for the fish shell.

## IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 project on GitHub](#)
- [The BlobFuse v1 setup documentation](#)

## Syntax

```
blobfuse2 completion fish --[flag-name]=[flag-value]
```

## Flags (options)

Flags that apply to `blobfuse2 completion fish` are inherited from the grandparent command, `blobfuse2`, or apply only to the `blobfuse2 completion` subcommands.

### Flags inherited from the BlobFuse2 command

The following flags are inherited from grandparent command `blobfuse2`:

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
disable-version-check		boolean	false	Enables or disables automatic version checking of the BlobFuse2 binaries
help	-h	n/a		Help info for the blobfuse2 command and subcommands

### Flags that apply to the BlobFuse2 completion subcommands

The following flags apply only to the `blobfuse2 completion` subcommands:

FLAG	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
no-descriptions	boolean	false	Disable completion descriptions

## Usage

To load completions in your current shell session:

```
blobfuse2 completion fish | source
```

To load completions for every new session, execute once:

```
blobfuse2 completion fish > ~/.config/fish/completions/blobfuse2.fish
```

**NOTE**

You will need to start a new shell for this setup to take effect.

## See also

- [The Blobfuse2 completion command \(preview\)](#)
- [The Blobfuse2 completion bash command \(preview\)](#)
- [The Blobfuse2 completion PowerShell command \(preview\)](#)
- [The Blobfuse2 completion zsh command \(preview\)](#)

# BlobFuse2 completion powershell command (preview)

8/22/2022 • 2 minutes to read • [Edit Online](#)

Use the `blobfuse2 completion powershell` command to generate the autocompletion script for BlobFuse2 for PowerShell.

## IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 project on GitHub](#)
- [The BlobFuse v1 setup documentation](#)

## Syntax

```
blobfuse2 completion powershell --[flag-name]=[flag-value]
```

## Flags (options)

Flags that apply to `blobfuse2 completion powershell` are inherited from the grandparent command, `blobfuse2`, or apply only to the `blobfuse2 completion` subcommands.

### Flags inherited from the BlobFuse2 command

The following flags are inherited from grandparent command `blobfuse2`:

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
disable-version-check		boolean	false	Enables or disables automatic version checking of the BlobFuse2 binaries
help	-h	n/a		Help info for the blobfuse2 command and subcommands

### Flags that apply to the BlobFuse2 completion subcommands

The following flags apply only to the `blobfuse2 completion` subcommands:

FLAG	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
no-descriptions	boolean	false	Disable completion descriptions

# Usage

To load completions in your current PowerShell session:

```
blobfuse2 completion powershell | Out-String | Invoke-Expression
```

To load completions for every new session, add the output of the above command to your PowerShell profile.

**NOTE**

You will need to start a new shell for this setup to take effect.

## See also

- [The Blobfuse2 completion command \(preview\)](#)
- [The Blobfuse2 completion bash command \(preview\)](#)
- [The Blobfuse2 completion fish command \(preview\)](#)
- [The Blobfuse2 completion zsh command \(preview\)](#)

# BlobFuse2 completion zsh command (preview)

8/22/2022 • 2 minutes to read • [Edit Online](#)

Use the `blobfuse2 completion zsh` command to generate the autocompletion script for BlobFuse2 for the zsh shell.

## IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 project on GitHub](#)
- [The BlobFuse v1 setup documentation](#)

## Syntax

```
blobfuse2 completion zsh --[flag-name]=[flag-value]
```

## Flags (options)

Flags that apply to `blobfuse2 completion zsh` are inherited from the grandparent command, `blobfuse2`, or apply only to the `blobfuse2 completion` subcommands.

### Flags inherited from the BlobFuse2 command

The following flags are inherited from grandparent command `blobfuse2`:

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
disable-version-check		boolean	false	Enables or disables automatic version checking of the BlobFuse2 binaries
help	-h	n/a		Help info for the blobfuse2 command and subcommands

### Flags that apply to the BlobFuse2 completion subcommands

The following flags apply only to the `blobfuse2 completion` subcommands:

FLAG	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
no-descriptions	boolean	false	Disable completion descriptions

## Usage

If shell completion is not already enabled in your environment you will need to enable it. To do so, run the

following command once:

```
echo "autoload -U compinit; compinit" >> ~/.zshrc
```

To load completions in your current shell session:

```
source <(blobfuse2 completion zsh); compdef _blobfuse2 blobfuse2
```

To load completions for every new session, execute once:

- On Linux:

```
blobfuse2 completion zsh > "${fpath[1]}/_blobfuse2"
```

- On macOS:

```
blobfuse2 completion zsh > /usr/local/share/zsh/site-functions/_blobfuse2
```

#### NOTE

You will need to start a new shell for this setup to take effect.

## See also

- [The Blobfuse2 completion command \(preview\)](#)
- [The Blobfuse2 completion bash command \(preview\)](#)
- [The Blobfuse2 completion fish command \(preview\)](#)
- [The Blobfuse2 completion PowerShell command \(preview\)](#)

# How to use the BlobFuse2 secure command (preview)

8/22/2022 • 2 minutes to read • [Edit Online](#)

Use the `blobfuse2 secure` command to encrypt, decrypt, or access settings in a BlobFuse2 configuration file.

## IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

If you need to use BlobFuse in a production environment, BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 setup documentation](#)
- [The BlobFuse v1 project on GitHub](#)

## Command Syntax

```
blobfuse2 secure [command] --[flag-name]=[flag-value]
```

## Arguments

[command]

The supported subcommands for `blobfuse2 secure` are:

COMMAND	DESCRIPTION
<a href="#">encrypt</a>	Encrypts a BlobFuse2 configuration file
<a href="#">decrypt</a>	Decrypts a BlobFuse2 configuration file
<a href="#">get</a>	Gets a specified value from an encrypted BlobFuse2 configuration file
<a href="#">set</a>	Sets a specified value in an encrypted BlobFuse2 configuration file

Select one of the command links in the table above to view the documentation for the individual subcommands, including the arguments and flags they support.

## Flags (options)

Flags that apply to `blobfuse2 secure` are inherited from the parent command, `blobfuse2`, or apply only to the `blobfuse2 secure` subcommands.

### Flags inherited from the BlobFuse2 command

The following flags are inherited from parent command `blobfuse2`:

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
disable-version-check		boolean	false	Enables or disables automatic version checking of the BlobFuse2 binaries
help	-h	n/a	n/a	Help info for the blobfuse2 command and subcommands

### Flags that apply only to the BlobFuse2 secure command

The following flags apply only to command `blobfuse2 secure` :

FLAG	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
config-file	string	./config.yaml	The path configuration file
output-file	string		Path and name for the output file
passphrase	string		The Key to be used for encryption or decryption Can also be specified by environment variable <code>BLOBFUSE2_SECURE_CONFIG_PASSPHRASE</code> . The key must be 16 (AES-128), 24 (AES-192), or 32 (AES-256) bytes in length.

## Examples

For examples, see the documentation for [the individual subcommands](#).

## See also

- [The Blobfuse2 secure encrypt command \(preview\)](#)
- [The Blobfuse2 secure decrypt command \(preview\)](#)
- [The Blobfuse2 secure get command \(preview\)](#)
- [The Blobfuse2 secure set command \(preview\)](#)

# How to use the BlobFuse2 secure encrypt command to encrypt a BlobFuse2 configuration file (preview)

8/22/2022 • 2 minutes to read • [Edit Online](#)

Use the `BlobFuse2 secure encrypt` command to encrypt a BlobFuse2 configuration file.

## IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

If you need to use BlobFuse in a production environment, BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 setup documentation](#)
- [The BlobFuse v1 project on GitHub](#)

## Syntax

```
blobfuse2 secure encrypt --[flag-name]=[flag-value]
```

## Flags (options)

Flags that apply to `blobfuse2 secure encrypt` are inherited from the grandparent command, `blobfuse2`, or apply only to the `blobfuse2 secure` subcommands.

### Flags inherited from the BlobFuse2 command

The following flags are inherited from grandparent command `blobfuse2`:

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
disable-version-check		boolean	false	Enables or disables automatic version checking of the BlobFuse2 binaries
help	-h	n/a		Help info for the blobfuse2 command and subcommands

### Flags inherited from the BlobFuse2 secure command

The following flags are inherited from parent command `blobfuse2 secure`:

FLAG	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
config-file	string	./config.yaml	The path configuration file
output-file	string		Path and name for the output file

FLAG	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
passphrase	string		The Key to be used for encryption or decryption Can also be specified by environment variable BLOBFUSE2_SECURE_CONFIG_PASSPHRASE. The key must be 16 (AES-128), 24 (AES-192), or 32 (AES-256) bytes in length.

## Examples

**NOTE**

The following examples assume you have already created a configuration file in the current directory.

Encrypt a BlobFuse2 configuration file using a passphrase:

```
blobfuse2 secure encrypt --config-file=./config.yaml --passphrase=PASSPHRASE
```

## See also

- [The Blobfuse2 secure decrypt command \(preview\)](#)
- [The Blobfuse2 secure get command \(preview\)](#)
- [The Blobfuse2 secure set command \(preview\)](#)
- [The Blobfuse2 secure command \(preview\)](#)

# How to use the BlobFuse2 secure decrypt command to decrypt a BlobFuse2 configuration file (preview)

8/22/2022 • 2 minutes to read • [Edit Online](#)

Use the `BlobFuse2 secure decrypt` command to decrypt a BlobFuse2 configuration file.

## IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

If you need to use BlobFuse in a production environment, BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 setup documentation](#)
- [The BlobFuse v1 project on GitHub](#)

## Syntax

```
blobfuse2 secure decrypt --[flag-name]=[flag-value]
```

## Flags (options)

Flags that apply to `blobfuse2 secure decrypt` are inherited from the grandparent command, `blobfuse2`, or apply only to the `blobfuse2 secure` subcommands.

### Flags inherited from the BlobFuse2 command

The following flags are inherited from grandparent command `blobfuse2`:

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
disable-version-check		boolean	false	Enables or disables automatic version checking of the BlobFuse2 binaries
help	-h	n/a		Help info for the blobfuse2 command and subcommands

### Flags inherited from the BlobFuse2 secure command

The following flags are inherited from parent command `blobfuse2 secure`:

FLAG	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
config-file	string	./config.yaml	The path configuration file
output-file	string		Path and name for the output file

FLAG	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
passphrase	string		The Key to be used for encryption or decryption Can also be specified by environment variable BLOBFUSE2_SECURE_CONFIG_PASSPHRASE. The key must be 16 (AES-128), 24 (AES-192), or 32 (AES-256) bytes in length.

## Examples

**NOTE**

The following examples assume you have already created a configuration file in the current directory.

Decrypt a BlobFuse2 configuration file using a passphrase:

```
blobfuse2 secure decrypt --config-file=./config.yaml --passphrase=PASSPHRASE
```

## See also

- [The Blobfuse2 secure encrypt command \(preview\)](#)
- [The Blobfuse2 secure get command \(preview\)](#)
- [The Blobfuse2 secure set command \(preview\)](#)
- [The Blobfuse2 secure command \(preview\)](#)

# How to use the BlobFuse2 secure get command to display the value of a parameter from an encrypted BlobFuse2 configuration file (preview)

8/22/2022 • 2 minutes to read • [Edit Online](#)

Use the `BlobFuse2 secure get` command to display the value of a specified parameter from an encrypted BlobFuse2 configuration file.

## IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

If you need to use BlobFuse in a production environment, BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 setup documentation](#)
- [The BlobFuse v1 project on GitHub](#)

## Syntax

```
blobfuse2 secure get --[flag-name]=[flag-value]
```

## Flags (options)

Flags that apply to `blobfuse2 secure get` are inherited from the grandparent command, `blobfuse2`, or apply only to the `blobfuse2 secure` subcommands.

### Flags inherited from the BlobFuse2 command

The following flags are inherited from grandparent command `blobfuse2`:

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
disable-version-check		boolean	false	Enables or disables automatic version checking of the BlobFuse2 binaries
help	-h	n/a		Help info for the blobfuse2 command and subcommands

### Flags inherited from the BlobFuse2 secure command

The following flags are inherited from parent command `blobfuse2 secure`:

FLAG	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
config-file	string	./config.yaml	The path configuration file

FLAG	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
output-file	string		Path and name for the output file
passphrase	string		The Key to be used for encryption or decryption Can also be specified by environment variable BLOBFUSE2_SECURE_CONFIG_PASSPHRASE. The key must be 16 (AES-128), 24 (AES-192), or 32 (AES-256) bytes in length.

### Flags that apply only to the BlobFuse2 secure get command

The following flags apply only to command `blobfuse2 secure get` command:

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
key		string		Configuration key (parameter) to be searched in an encrypted config file

## Examples

### NOTE

The following examples assume you have already created a configuration file in the current directory.

Get the value of parameter `logging.log_level` from an encrypted BlobFuse2 configuration file using a passphrase:

```
blobfuse2 secure get --config-file=~/config.yaml --passphrase=PASSPHRASE --key=logging.log_level
```

## See also

- [The Blobfuse2 secure set command \(preview\)](#)
- [The Blobfuse2 secure encrypt command \(preview\)](#)
- [The Blobfuse2 secure decrypt command \(preview\)](#)
- [The Blobfuse2 secure command \(preview\)](#)

# How to use the BlobFuse2 secure set command to change the value of a parameter in an encrypted BlobFuse2 configuration file (preview)

8/22/2022 • 2 minutes to read • [Edit Online](#)

Use the `BlobFuse2 secure set` command to change the value of a specified parameter in an encrypted BlobFuse2 configuration file.

## IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

If you need to use BlobFuse in a production environment, BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 setup documentation](#)
- [The BlobFuse v1 project on GitHub](#)

## Syntax

```
blobfuse2 secure set --[flag-name]=[flag-value]
```

## Flags (options)

Flags that apply to `blobfuse2 secure set` are inherited from the grandparent command, `blobfuse2`, or apply only to the `blobfuse2 secure` subcommands.

### Flags inherited from the BlobFuse2 command

The following flags are inherited from grandparent command `blobfuse2`:

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
disable-version-check		boolean	false	Enables or disables automatic version checking of the BlobFuse2 binaries
help	-h	n/a		Help info for the blobfuse2 command and subcommands

### Flags inherited from the BlobFuse2 secure command

The following flags are inherited from parent command `blobfuse2 secure`:

FLAG	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
config-file	string	./config.yaml	The path configuration file

FLAG	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
output-file	string		Path and name for the output file
passphrase	string		The Key to be used for encryption or decryption Can also be specified by environment variable BLOBFUSE2_SECURE_CONFIG_PASSPHRASE. The key must be 16 (AES-128), 24 (AES-192), or 32 (AES-256) bytes in length.

## Flags that apply only to the BlobFuse2 secure set command

The following flags apply only to command `blobfuse2 secure set` command:

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
key		string		Configuration key (parameter) to be updated in an encrypted config file
value		string		New value for the configuration key (parameter) to be updated in an encrypted config file

## Examples

### NOTE

The following examples assume you have already created a configuration file in the current directory.

Set the value of parameter `logging.log_level` in an encrypted BlobFuse2 configuration file to "log\_debug":

```
blobfuse2 secure set --config-file=config.yaml --passphrase=PASSPHRASE --key=logging.log_level --
value=log_debug
```

## See also

- [The Blobfuse2 secure get command \(preview\)](#)
- [The Blobfuse2 secure encrypt command \(preview\)](#)
- [The Blobfuse2 secure decrypt command \(preview\)](#)
- [The Blobfuse2 secure command \(preview\)](#)

# BlobFuse2 version command (preview)

8/22/2022 • 2 minutes to read • [Edit Online](#)

Use the `blobfuse2 version` command to display the current version of BlobFuse2, and optionally check for latest version.

## IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 project on GitHub](#)
- [The BlobFuse v1 setup documentation](#)

## Syntax

```
blobfuse2 version --[flag-name]=[flag-value]
```

## Flags (options)

Some flags are inherited from the parent command, `blobfuse2`, and others only apply to the `blobfuse2 version` command.

### Flags inherited from the BlobFuse2 command

The following flags are inherited from parent command `blobfuse2`):

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
disable-version-check		boolean	false	Enables or disables automatic version checking of the BlobFuse2 binaries
help	-h	n/a	n/a	Help info for the blobfuse2 command and subcommands

### Flags that apply only to the BlobFuse2 version command

The following flags apply only to command `blobfuse2 version`:

FLAG	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
check	boolean	false	Check for the latest version

## Examples

```
blobfuse2 version --check=true
```

## See also

- [The Blobfuse2 command set \(preview\)](#)

# BlobFuse2 help command (preview)

8/22/2022 • 2 minutes to read • [Edit Online](#)

Use the `blobfuse2 help` command to get help info for the BlobFuse2 command and subcommands.

## IMPORTANT

BlobFuse2 is the next generation of BlobFuse and is currently in preview. This preview version is provided without a service level agreement, and is not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

BlobFuse v1 is generally available (GA). For information about the GA version, see:

- [The BlobFuse v1 project on GitHub](#)
- [The BlobFuse v1 setup documentation](#)

## Syntax

The `blobfuse2 help` command has 2 formats:

```
blobfuse2 help --[flag-name]=[flag-value]
```

```
blobfuse2 help [command] --[flag-name]=[flag-value]
```

## Arguments

```
[command]
```

You can get help information for any of the specific BlobFuse2 commands. The supported `blobfuse2` commands are:

COMMAND	DESCRIPTION
<a href="#">mount</a>	Mounts blob storage containers and displays existing mount points
<a href="#">unmount</a>	Unmounts previously mounted blob storage containers
<a href="#">mountv1</a>	Generates a configuration file for BlobFuse2 from a BlobFuse v1 configuration file
<a href="#">completion</a>	Generates the autocompletion script for BlobFuse2 for a specified shell
<a href="#">secure</a>	Encrypts, decrypts, or accesses settings in a BlobFuse2 configuration file
<a href="#">version</a>	Displays the current version of BlobFuse2, and optionally checks for the latest version

Select one of the command links in the table above to view the documentation for the individual commands, including the arguments and flags they support.

# Flags (options)

The flags available for `blobfuse2 help` are inherited from the parent command, `blobfuse2`.

## Flags inherited from the BlobFuse2 command

The following flags are inherited from parent command `blobfuse2`):

FLAG	SHORT VERSION	VALUE TYPE	DEFAULT VALUE	DESCRIPTION
disable-version-check		boolean	false	Enables or disables automatic version checking of the BlobFuse2 binaries
help	-h	n/a	n/a	Help info for the blobfuse2 command and subcommands

## Examples

Get general BlobFuse2 help:

```
blobfuse2 help
```

Get help for the `blobfuse2 mount` command:

```
blobfuse2 help mount
```

Get help for the `blobfuse2 secure encrypt` subcommand:

```
blobfuse2 help secure encrypt
```

## See also

- [What is Blobfuse2? \(preview\)](#)
- [The Blobfuse2 command set \(preview\)](#)

# Azure Blob Storage monitoring data reference

8/22/2022 • 13 minutes to read • [Edit Online](#)

See [Monitoring Azure Storage](#) for details on collecting and analyzing monitoring data for Azure Storage.

## Metrics

The following tables list the platform metrics collected for Azure Storage.

### Capacity metrics

Capacity metrics values are refreshed daily (up to 24 Hours). The time grain defines the time interval for which metrics values are presented. The supported time grain for all capacity metrics is one hour (PT1H).

Azure Storage provides the following capacity metrics in Azure Monitor.

#### Account Level

This table shows [account-level metrics](#).

METRIC	DESCRIPTION
UsedCapacity	<p>The amount of storage used by the storage account. For standard storage accounts, it's the sum of capacity used by blob, table, file, and queue. For premium storage accounts and Blob storage accounts, it is the same as BlobCapacity.</p> <p>Unit: Bytes Aggregation Type: Average Value example: 1024</p>

#### Blob storage

This table shows [Blob storage metrics](#).

METRIC	DESCRIPTION
BlobCapacity	<p>The total of Blob storage used in the storage account.</p> <p>Unit: Bytes Aggregation Type: Average Value example: 1024 Dimensions: <a href="#">BlobType</a>, and <a href="#">Tier</a> (<a href="#">Definition</a>)</p>
BlobCount	<p>The number of blob objects stored in the storage account.</p> <p>Unit: Count Aggregation Type: Average Value example: 1024 Dimensions: <a href="#">BlobType</a>, and <a href="#">Tier</a> (<a href="#">Definition</a>)</p>
BlobProvisionedSize	<p>The amount of storage provisioned in the storage account. This metric is applicable to premium storage accounts only.</p> <p>Unit: bytes Aggregation Type: Average</p>
ContainerCount	<p>The number of containers in the storage account.</p> <p>Unit: Count Aggregation Type: Average Value example: 1024</p>
IndexCapacity	<p>The amount of storage used by ADLS Gen2 Hierarchical Index</p> <p>Unit: Bytes Aggregation Type: Average Value example: 1024</p>

### Transaction metrics

Transaction metrics are emitted on every request to a storage account from Azure Storage to Azure Monitor. In the case of no activity on your storage account, there will be no data on transaction metrics in the period. All transaction metrics are available at both account and Blob storage service level. The time grain defines the time interval that metric values are presented. The supported time grains for all transaction metrics are PT1H and PT1M.

Azure Storage provides the following transaction metrics in Azure Monitor.

METRIC	DESCRIPTION
Transactions	<p>The number of requests made to a storage service or the specified API operation. This number includes successful and failed requests, as well as requests that produced errors.</p> <p>Unit: Count Aggregation Type: Total Applicable dimensions: <a href="#">ResponseType</a>, <a href="#">GeoType</a>, <a href="#">ApiName</a>, and <a href="#">Authentication</a> (<a href="#">Definition</a>) Value example: 1024</p>

METRIC	DESCRIPTION
Ingress	<p>The amount of ingress data. This number includes ingress from an external client into Azure Storage as well as ingress within Azure.</p> <p>Unit: Bytes Aggregation Type: Total Applicable dimensions: GeoType, ApiName, and Authentication (<a href="#">Definition</a>) Value example: 1024</p>
Egress	<p>The amount of egress data. This number includes egress from an external client into Azure Storage as well as egress within Azure. As a result, this number does not reflect billable egress.</p> <p>Unit: Bytes Aggregation Type: Total Applicable dimensions: GeoType, ApiName, and Authentication (<a href="#">Definition</a>) Value example: 1024</p>
SuccessServerLatency	<p>The average time used to process a successful request by Azure Storage. This value does not include the network latency specified in SuccessE2ELatency.</p> <p>Unit: Milliseconds Aggregation Type: Average Applicable dimensions: GeoType, ApiName, and Authentication (<a href="#">Definition</a>) Value example: 1024</p>
SuccessE2ELatency	<p>The average end-to-end latency of successful requests made to a storage service or the specified API operation. This value includes the required processing time within Azure Storage to read the request, send the response, and receive acknowledgment of the response. The difference between SuccessE2ELatency and SuccessServerLatency values is the latency likely caused by the network and the client.</p> <p>Unit: Milliseconds Aggregation Type: Average Applicable dimensions: GeoType, ApiName, and Authentication (<a href="#">Definition</a>) Value example: 1024</p>
Availability	<p>The percentage of availability for the storage service or the specified API operation. Availability is calculated by taking the total billable requests value and dividing it by the number of applicable requests, including those requests that produced unexpected errors. All unexpected errors result in reduced availability for the storage service or the specified API operation.</p> <p>Unit: Percent Aggregation Type: Average Applicable dimensions: GeoType, ApiName, and Authentication (<a href="#">Definition</a>) Value example: 99.99</p>

## Metrics dimensions

Azure Storage supports following dimensions for metrics in Azure Monitor.

### Dimensions available to all storage services

DIMENSION NAME	DESCRIPTION
GeoType	Transaction from Primary or Secondary cluster. The available values include Primary and Secondary. It applies to Read Access Geo Redundant Storage(RA-GRS) when reading objects from secondary tenant.

DIMENSION NAME	DESCRIPTION
ResponseType	<p>Transaction response type. The available values include:</p> <ul style="list-style-type: none"> <li>• <b>ServerOtherError</b>: All other server-side errors except described ones</li> <li>• <b>ServerBusyError</b>: Authenticated request that returned an HTTP 503 status code.</li> <li>• <b>ServerTimeoutError</b>: Timed-out authenticated request that returned an HTTP 500 status code. The timeout occurred due to a server error.</li> <li>• <b>AuthenticationError</b>: The request failed to be authenticated by the server.</li> <li>• <b>AuthorizationError</b>: Authenticated request that failed due to unauthorized access of data or an authorization failure.</li> <li>• <b>NetworkError</b>: Authenticated request that failed due to network errors. Most commonly occurs when a client prematurely closes a connection before timeout expiration.</li> <li>• <b>ClientAccountBandwidthThrottlingError</b>: The request is throttled on bandwidth for exceeding <a href="#">storage account scalability limits</a>.</li> <li>• <b>ClientAccountRequestThrottlingError</b>: The request is throttled on request rate for exceeding <a href="#">storage account scalability limits</a>.</li> <li>• <b>ClientThrottlingError</b>: Other client-side throttling error. ClientAccountBandwidthThrottlingError and ClientAccountRequestThrottlingError are excluded.</li> <li>• <b>ClientShareEgressThrottlingError</b>: Applicable to premium files shares only. Other client-side throttling error. The request failed due to egress bandwidth throttling for exceeding share limits. ClientAccountBandwidthThrottlingError is excluded.</li> <li>• <b>ClientShareIngressThrottlingError</b>: Applicable to premium files shares only. Other client-side throttling error. The request failed due to ingress bandwidth throttling for exceeding share limits. ClientAccountBandwidthThrottlingError is excluded.</li> <li>• <b>ClientShareOpsThrottlingError</b>: Other client-side throttling error. The request failed due to IOPS throttling. ClientAccountRequestThrottlingError is excluded.</li> <li>• <b>ClientTimeoutError</b>: Timed-out authenticated request that returned an HTTP 500 status code. If the client's network timeout or the request timeout is set to a lower value than expected by the storage service, it is an expected timeout. Otherwise, it is reported as a ServerTimeoutError.</li> <li>• <b>ClientOtherError</b>: All other client-side errors except described ones.</li> <li>• <b>Success</b>: Successful request</li> <li>• <b>SuccessWithThrottling</b>: Successful request when a SMB client gets throttled in the first attempt(s) but succeeds after retries.</li> <li>• <b>SuccessWithShareEgressThrottling</b>: Applicable to premium files shares only. Successful request when a SMB client gets throttled due to egress bandwidth throttling in the first attempt(s) but succeeds after retries.</li> <li>• <b>SuccessWithShareIngressThrottling</b>: Applicable to premium files shares only. Successful request when a SMB client gets throttled due to ingress bandwidth throttling in the first attempt(s) but succeeds after retries.</li> <li>• <b>SuccessWithShareOpsThrottling</b>: Successful request when a SMB client gets throttled due to IOPS throttling in the first attempt(s) but succeeds after retries.</li> </ul>
ApiName	The name of operation. If a failure occurs before the name of the <a href="#">operation</a> is identified, the name appears as "Unknown". You can use the value of the <b>ResponseType</b> dimension to learn more about the failure.
Authentication	<p>Authentication type used in transactions. The available values include:</p> <ul style="list-style-type: none"> <li>• <b>AccountKey</b>: The transaction is authenticated with storage account key.</li> <li>• <b>SAS</b>: The transaction is authenticated with shared access signatures.</li> <li>• <b>OAuth</b>: The transaction is authenticated with OAuth access tokens.</li> <li>• <b>Anonymous</b>: The transaction is requested anonymously. It doesn't include preflight requests.</li> <li>• <b>AnonymousPreflight</b>: The transaction is preflight request.</li> </ul>
TransactionType	<p>Type of transaction. The available values include:</p> <ul style="list-style-type: none"> <li>• <b>User</b>: The transaction was made by customer.</li> <li>• <b>System</b>: The transaction was made by system process.</li> </ul>

#### Dimensions specific to Blob storage

DIMENSION NAME	DESCRIPTION
BlobType	The type of blob for Blob metrics only. The supported values are <b>BlockBlob</b> , <b>PageBlob</b> , and <b>Azure Data Lake Storage</b> . Append blobs are included in <b>BlockBlob</b> .

DIMENSION NAME	DESCRIPTION
Tier	Azure storage offers different access tiers, which allow you to store blob object data in the most cost-effective manner. See more in <a href="#">Azure Storage blob tier</a> . The supported values include: <ul style="list-style-type: none"> <li>• Hot: Hot tier</li> <li>• Cool: Cool tier</li> <li>• Archive: Archive tier</li> <li>• Premium: Premium tier for block blob</li> <li>• P4/P6/P10/P15/P20/P30/P40/P50/P60: Tier types for premium page blob</li> <li>• Standard: Tier type for standard page Blob</li> <li>• Untiered: Tier type for general purpose v1 storage account</li> </ul>

For the metrics supporting dimensions, you need to specify the dimension value to see the corresponding metrics values. For example, if you look at **Transactions** value for successful responses, you need to filter the **ResponseType** dimension with **Success**. If you look at **BlobCount** value for Block Blob, you need to filter the **BlobType** dimension with **BlockBlob**.

## Resource logs

The following table lists the properties for Azure Storage resource logs when they're collected in Azure Monitor Logs or Azure Storage. The properties describe the operation, the service, and the type of authorization that was used to perform the operation.

### Fields that describe the operation

```
{
 "time": "2019-02-28T19:10:21.2123117Z",
 "resourceId": "/subscriptions/12345678-2222-3333-4444-
555555555555/resourceGroups/mytestgrp/providers/Microsoft.Storage/storageAccounts/testaccount1/blobServices/d
efault",
 "category": "StorageWrite",
 "operationName": "PutBlob",
 "operationVersion": "2017-04-17",
 "schemaVersion": "1.0",
 "statusCode": 201,
 "statusText": "Success",
 "durationMs": 5,
 "callerIpAddress": "192.168.0.1:11111",
 "correlationId": "ad881411-201e-004e-1c99-cfd67d000000",
 "location": "uswestcentral",
 "uri": "http://mystorageaccount.blob.core.windows.net/cont1/blobname?timeout=10"
}
```

PROPERTY	DESCRIPTION
time	The Universal Time Coordinated (UTC) time when the request was received by storage. For example: <code>2018/11/08 21:09:36.6900118</code> .
resourceId	The resource ID of the storage account. For example: <code>/subscriptions/208841be-a4v3-4234-9450- 08b90c0f4/resourceGroups/ myresourcegroup/providers/Microsoft.Storage/storageAccounts/mystorageaccount/storageAccounts/blobSer</code>
category	The category of the requested operation. For example: <code>StorageRead</code> , <code>StorageWrite</code> , or <code>StorageDelete</code> .
operationName	The type of REST operation that was performed. For a complete list of operations, see <a href="#">Storage Analytics Logged Operations and Status Messages topic</a> .
operationVersion	The storage service version that was specified when the request was made. This is equivalent to the value of the <code>x-ms-version</code> header. For example: <code>2017-04-17</code> .
schemaVersion	The schema version of the log. For example: <code>1.0</code> .
statusCode	The HTTP or <a href="#">SMB</a> status code for the request. If the HTTP request is interrupted, this value might be set to <code>Unknown</code> . For example: <code>206</code> .
statusText	The status of the requested operation. For a complete list of status messages, see <a href="#">Storage Analytics Logged Operations and Status Messages topic</a> . In version 2017-04-17 and later, the status message <code>ClientOtherError</code> isn't used. Instead, this field contains an error code. For example: <code>SASSuccess</code> .
durationMs	The total time, expressed in milliseconds, to perform the requested operation. This includes the time to read the incoming request, and to send the response to the requester. For example: <code>12</code> .
callerIpAddress	The IP address of the requester, including the port number. For example: <code>192.100.0.102:4362</code> .
correlationId	The ID that is used to correlate logs across resources. For example: <code>b99ba45e-a01e-0042-4ea6-772bbb000000</code> .

PROPERTY	DESCRIPTION
location	The location of storage account. For example: North Europe
protocol	The protocol that is used in the operation. For example: HTTP, HTTPS, SMB, or NFS
uri	Uniform resource identifier that is requested.

**Fields that describe how the operation was authenticated**

```
{
 "identity": {
 "authorization": [
 {
 "action": "Microsoft.Storage/storageAccounts/blobServices/containers/blobs/read",
 " principals": [
 {
 "id": "fde5ba15-4355-4223-b811-cccccccccc",
 "type": "User"
 }
],
 "roleAssignmentId": "ecf75cb8-491c-4a25-ad6e-aaaaaaaaaa",
 "roleDefinitionId": "b7e6dc6d-f1e8-4753-8033-ffffffffffff"
 }
],
 "requester": {
 "appId": "6914589-1327-4635-9f55-bbbbbbbbbb",
 "audience": "https://storage.azure.com/",
 "objectId": "fde5ba15-4355-4223-b811-cccccccccc",
 "tenantId": "72f988bf-86f1-41af-91ab-ddddddd",
 "tokenIssuer": "https://sts.windows.net/72f988bf-86f1-41af-91ab-eeeeeeeeee/"
 },
 "type": "OAuth"
 },
}
```

PROPERTY	DESCRIPTION
identity / type	The type of authentication that was used to make the request. For example: OAuth, Kerberos, SAS Key, Account Key, or Anonymous
identity / tokenHash	The SHA-256 hash of the authentication token used on the request. When the authentication type is Account Key, the format is "key1   key2 (SHA256 hash of the key)". For example: key1(5RTE343A6FEB12342672AFD40072B70D4A91BGH5CDF797EC56BF82B2C3635CE)  When authentication type is SAS Key, the format is "key1   key2 (SHA 256 hash of the key),SasSignature(SHA 256 hash of the SAS token)". For example: key1(0A0XEB8AAD4354H19722ED12342443F0DC8FAF3E6GF8C8AD805DE6D563E0E5F8A),SasSignature(04D64C2B3A704145 When authentication type is OAuth, the format is "SHA 256 hash of the OAuth token". For example: B3CC905C64B3351573D806751312317FE4E910877E7CBAFA9D95E0BE923DD25C For other authentication types, there is no tokenHash field.
authorization / action	The action that is assigned to the request.
authorization / roleAssignmentId	The role assignment ID. For example: 4e2521b7-13be-4363-aeda-111111111111
authorization / roleDefinitionId	The role definition ID. For example: ba92f5b4-2d11-453d-a403-111111111111
principals / id	The ID of the security principal. For example: a4711f3a-254f-4cfb-8a2d-111111111111
principals / type	The type of security principal. For example: ServicePrincipal
requester / appId	The Open Authorization (OAuth) application ID that is used as the requester. For example: d3f7d5fe-e64a-4e4e-871d-333333333333
requester / audience	The OAuth audience of the request. For example: https://storage.azure.com
requester / objectId	The OAuth object ID of the requester. In case of Kerberos authentication, represents the object identifier of Kerberos authenticated user. For example: 0e0bf547-55e5-465c-91b7-2873712b249c
requester / tenantId	The OAuth tenant ID of identity. For example: 72f988bf-86f1-41af-91ab-222222222222
requester / tokenIssuer	The OAuth token issuer. For example: https://sts.windows.net/72f988bf-86f1-41af-91ab-222222222222

PROPERTY	DESCRIPTION
requester / upn	The User Principal Name (UPN) of requestor. For example: <code>someone@contoso.com</code> .
requester / userName	This field is reserved for internal use only.

#### Fields that describe the service

```
{
 "properties": {
 "accountName": "testaccount1",
 "requestUrl": "https://testaccount1.blob.core.windows.net:443/upload?
restype=container&comp=list&prefix=&delimiter=%2F&marker=&maxresults=30&include=metadata&_=1551405598426",
 "userAgentHeader": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/64.0.3282.140 Safari/537.36 Edge/17.17134",
 "referrerHeader": "blob:https://portal.azure.com/6f50025f-3b88-488d-b29e-3c592a31ddc9",
 "clientRequestId": "",
 "etag": "",
 "serverLatencyMs": 63,
 "serviceType": "blob",
 "operationCount": 0,
 "requestHeaderSize": 2658,
 "requestBodySize": 0,
 "responseHeaderSize": 295,
 "responseBodySize": 2018,
 "contentLengthHeader": 0,
 "requestMd5": "",
 "serverMd5": "",
 "lastModifiedTime": "",
 "conditionsUsed": "",
 "smbTreeConnectID" : "0x3",
 "smbPersistentHandleID" : "0x6003f",
 "smbVolatileHandleID" : "0xFFFFFFF00000065",
 "smbMessageID" : "0x3b165",
 "smbCreditsConsumed" : "0x3",
 "smbCommandDetail" : "0x2000 bytes at offset 0xf2000",
 "smbfileId" : "0x9223442405598953",
 "smbSessionID" : "0x8530280128000049",
 "smbCommandMajor" : "0x6",
 "smbCommandMinor" : "DirectoryCloseAndDelete"
 }
}
```

PROPERTY	DESCRIPTION
accountName	The name of the storage account. For example: <code>mystorageaccount</code> .
requestUrl	The URL that is requested.
userAgentHeader	The <b>User-Agent</b> header value, in quotes. For example: <code>WA-Storage/6.2.0 (.NET CLR 4.0.30319.42000; Win32NT 6.2.9200.0)</code> .
referrerHeader	The <b>Referrer</b> header value. For example: <code>http://contoso.com/about.html</code> .
clientRequestId	The <b>x-ms-client-request-id</b> header value of the request. For example: <code>360b66a6-ad4f-4c4a-84a4-0ad7cb44f7a6</code> .
etag	The Etag identifier for the returned object, in quotes. For example: <code>0x80101f7e48662c4</code> .
serverLatencyMs	The total time expressed in milliseconds to perform the requested operation. This value doesn't include network latency (the time to read the incoming request and send the response to the requester). For example: <code>22</code> .
serviceType	The service associated with this request. For example: <code>blob</code> , <code>table</code> , <code>files</code> , or <code>queue</code> .
operationCount	The number of each logged operation that is involved in the request. This count starts with an index of <code>0</code> . Some requests require more than one operation. Most requests perform only one operation. For example: <code>1</code> .
requestHeaderSize	The size of the request header expressed in bytes. For example: <code>578</code> . If a request is unsuccessful, this value might be empty.
requestBodySize	The size of the request packets, expressed in bytes, that are read by the storage service. For example: <code>0</code> . If a request is unsuccessful, this value might be empty.
responseHeaderSize	The size of the response header expressed in bytes. For example: <code>216</code> . If a request is unsuccessful, this value might be empty.
responseBodySize	The size of the response packets written by the storage service, in bytes. If a request is unsuccessful, this value may be empty. For example: <code>216</code> .

PROPERTY	DESCRIPTION
<b>requestMd5</b>	The value of either the <b>Content-MD5</b> header or the <b>x-ms-content-md5</b> header in the request. The MD5 hash value specified in this field represents the content in the request. For example: <code>788815fd0198be0d275ad329cafd1830</code> . This field can be empty.
<b>serverMd5</b>	The value of the MD5 hash calculated by the storage service. For example: <code>3228b3cf1069a5489b298446321f8521</code> . This field can be empty.
<b>lastModifiedTime</b>	The Last Modified Time (LMT) for the returned object. For example: <code>Tuesday, 09-Aug-11 21:13:26 GMT</code> . This field is empty for operations that can return multiple objects.
<b>conditionsUsed</b>	A semicolon-separated list of key-value pairs that represent a condition. The conditions can be any of the following: <ul style="list-style-type: none"> <li>• If-Modified-Since</li> <li>• If-Unmodified-Since</li> <li>• If-Match</li> <li>• If-None-Match</li> </ul> For example: <code>If-Modified-Since=Friday, 05-Aug-11 19:11:54 GMT</code> .
<b>contentLengthHeader</b>	The value of the Content-Length header for the request sent to the storage service. If the request was successful, this value is equal to <code>requestBodySize</code> . If a request is unsuccessful, this value may not be equal to <code>requestBodySize</code> , or it might be empty.
<b>tlsVersion</b>	The TLS version used in the connection of request. For example: <code>TLS 1.2</code> .
<b>smbTreeConnectID</b>	The Server Message Block (SMB) <code>treeConnectId</code> established at tree connect time. For example: <code>0x3</code> .
<b>smbPersistentHandleID</b>	Persistent handle ID from an SMB2 CREATE request that survives network reconnects. Referenced in <a href="#">MS-SMB2 2.2.14.1</a> as <code>SMB2_FILEID.Persistent</code> . For example: <code>0x6003f</code> .
<b>smbVolatileHandleID</b>	Volatile handle ID from an SMB2 CREATE request that is recycled on network reconnects. Referenced in <a href="#">MS-SMB2 2.2.14.1</a> as <code>SMB2_FILEID.Volatile</code> . For example: <code>0xFFFFFFFF00000065</code> .
<b>smbMessageID</b>	The connection relative <code>MessageId</code> . For example: <code>0x3b165</code> .
<b>smbCreditsConsumed</b>	The ingress or egress consumed by the request, in units of 64k. For example: <code>0x3</code> .
<b>smbCommandDetail</b>	More information about this specific request rather than the general type of request. For example: <code>0x2000 bytes at offset 0xf2000</code> .
<b>smbFileId</b>	The <code>FileId</code> associated with the file or directory. Roughly analogous to an NTFS FileId. For example: <code>0x9223442405598953</code> .
<b>smbSessionID</b>	The SMB2 <code>SessionId</code> established at session setup time. For example: <code>0x8530280128000049</code> .
<b>smbCommandMajor uint32</b>	Value in the <code>SMB2_HEADER.Command</code> . Currently, this is a number between 0 and 18 inclusive. For example: <code>0x6</code> .
<b>smbCommandMinor</b>	The subclass of <code>SmbCommandMajor</code> , where appropriate. For example: <code>DirectoryCloseAndDelete</code> .

## See also

- See [Monitoring Azure Storage](#) for a description of monitoring Azure Storage.
- See [Monitoring Azure resources with Azure Monitor](#) for details on monitoring Azure resources.

# Host keys for SSH File Transfer Protocol (SFTP) support for Azure Blob Storage (preview)

8/22/2022 • 6 minutes to read • [Edit Online](#)

This article contains a list of valid host keys used to connect to Azure Blob Storage from SFTP clients.

Blob storage now supports the SSH File Transfer Protocol (SFTP). This support provides the ability to securely connect to Blob Storage accounts via an SFTP endpoint, allowing you to leverage SFTP for file access, file transfer, as well as file management. For more information, see [SSH File Transfer Protocol \(SFTP\) support for Azure Blob Storage](#).

When you connect to Blob Storage by using an SFTP client, you might be prompted to trust a host key. During the public preview, you can verify the host key by finding that key in the list presented in this article.

## IMPORTANT

SFTP support is currently in PREVIEW and is available on general-purpose v2 and premium block blob accounts. Complete [this form](#) BEFORE using the feature in preview. Registration via 'preview features' is NOT required and confirmation email will NOT be sent after filling out the form. You can IMMEDIATELY access the feature.

After testing your end-to-end scenarios with SFTP please share your experience via [this form](#).

See the [Supplemental Terms of Use for Microsoft Azure Previews](#) for legal terms that apply to Azure features that are in beta, preview, or otherwise not yet released into general availability.

## Valid host keys

REGION	HOST KEY TYPE	SHA 256 FINGERPRINT <sup>1</sup>	PUBLIC KEY
West Europe	rsa-sha2-256	IeHrQ+N6wAdLMKSMsJiML4XqMrkf	AAAAAB3NzaC1yc2EAAAQABAAQDZL63ZKhrW1wN8gkPvq43uUh88n0V6Gw1TH2/sEpI
West Europe	rsa-sha2-512	7+VdJ21y+HcaNRZZea8tk1AjkCN	AAAAAB3NzaC1yc2EAAAQABAAQDYAmiv6Tk/o02McJi79d1IdPlu15FhfsdP1uycW+
West Europe	ecdsa-sha2-nistp256	0WNMhCNJE1YFBpHeAdU5h+Pfj	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBBANx85rJLXM8QZi33y8
West Europe	ecdsa-sha2-nistp384	90g+JFQChjb300V0YIGSVtKNotr	AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAIBmlzdHAzODQAAABhBNjgrlFy2zsyhNvXlw
East US	rsa-sha2-256	F6pNN5Py68/1hVRGEoCwpY5H7vwR	AAAAAB3NzaC1yc2EAAAQABAAQDQAIUb94zwLf0e/+0eiajE0X70d2nuqyLyAqp0b7n
East US	rsa-sha2-512	MIpoRIiCtEK123MN+S2bLqm5GKC1	AAAAAB3NzaC1yc2EAAAQABAAQ8C8Ut7Rq7Vak26f29czig5auq334N9mNaJmdtWoT3
East US	ecdsa-sha2-nistp256	ixDeCdmQB9RoqdJiVdXyFVsRqlr	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBNrdcVT12fftnDcJYL8
East US	ecdsa-sha2-nistp384	DPTC6EI0rsxzpGt6IZzAN67n1ZL	AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAIBmlzdHAzODQAAABhBEP3CuVPVNVnFuojR4
West India	rsa-sha2-256	Fkh7r/t0Jy1cZC6nI75Vs01s53ug	AAAAAB3NzaC1yc2EAAAQABAAQDHCzL151bbBLWK7tCxVx EhALQMzuYKEwoyS1/oCS
West India	rsa-sha2-512	xDtcfgfElRGUgwIu9tRmSQ58WEOK	AAAAAB3NzaC1yc2EAAAQABAAQCEhufp18nKehU4/GOWMkxJ87t22TyG5bNdVCP02A
West India	ecdsa-sha2-nistp256	t+PVPMsVEgQ3FPNploXz7m025PFj	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBCzR5dh3wfN5bRqlF
West India	ecdsa-sha2-nistp384	pLOdd+3JNeLVcPYyI0rSwomhMk	AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAIBmlzdHAzODQAAABhL2PEknfZpPAT4ejqBJ
East US 2	rsa-sha2-256	K+Q0glmdpev3bvEKUGBTiOGMxwTl	AAAAAB3NzaC1yc2EAAAQABAAQDOA2A1tIG/TUXoVFHo1tyw8qnC0tm2gq2NFTdfyD
East US 2	rsa-sha2-512	UKT1qPRfpm+yzpRMukKpBCRFn0d2	AAAAAB3NzaC1yc2EAAAQABAAQAC/HCjYc4tMVNkmbEDT0HXVhyKyzufrad8pvGb3bw
East US 2	ecdsa-sha2-nistp256	bouiC5HdtrUU19Rjbym8R94fbMc	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBBJshJi18IECu6neLras
East US 2	ecdsa-sha2-nistp384	vWnP1GaQ0Y4LFj9X5Q2qN/NMF92+	AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAIBmlzdHAzODQAAABhBByJNAb1wxCNVqedg5F
West US	rsa-sha2-256	kqxoK1j6vHU8o8Xai1V87tZFE9r	AAAAAB3NzaC1yc2EAAAQABAAQD7gh0mFw3irAKusX3ai60E0K0502CMlezbZ0AJE
West US	rsa-sha2-512	/PP9B/9KEa+Quyump1Yt05Lfk0L	AAAAAB3NzaC1yc2EAAAQABAAQC8R8bFe8QSTYKK+4evMpn1B8y0rQCqikTyviqd4rv
West US	ecdsa-sha2-nistp256	peqBfcwZRW4QzLi69HiCUIUtwdtf	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBBCTos/zmSn15kn1lk
West US	ecdsa-sha2-nistp384	sg63Cc3Mvn9hoapGaEuZByscUEM	AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAIBmlzdHAzODQAAABhBgzX2t9ALjFwpcBLm4B
East US 2 EUAP	rsa-sha2-256	dKP64wLSbRo1v2MV02TwH5wFPb	AAAAAB3NzaC1yc2EAAAQABAAQAC3PqLDkkqUxrZSabiEZsI6T1jYRh5cp+F5ktPCw7
East US 2 EUAP	rsa-sha2-512	M390fv6366yGPdeFZ0/2B7Ui6Jze	AAAAAB3NzaC1yc2EAAAQABAAQC+1NvYoRon15Tr2wNGmL+uR17GoVKwBsKFvhbRH
East US 2 EUAP	ecdsa-sha2-nistp256	X+c1NIpAJGvU31Uj3Vd20s4J7bc	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBBK+U6CE6con74cCntkF
East US 2 EUAP	ecdsa-sha2-nistp384	Q3zIFFO11UfCrMq6Eh7nP1/Vivgf	AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAIBmlzdHAzODQAAABhBDW RJ0+e8kZpalcdg7H
Australia Central	rsa-sha2-256	q2p0jwwgUuAMU3irD12D+sBh8wQf	AAAAAB3NzaC1yc2EAAAQABAAQDnqOrNklxmyreRye7N72y1BCensxxPTBWx/CfdbdbG

Region	Host Key Type	SHA 256 Fingerprint	Public Key
Australia Central	rsa-sha2-512	+tDlVAC4I+7DhQn9JguFBPu0/Hdt	AAAAB3NzaC1yc2EAAADAQABAAQCb0ETMwpU8w7uwu1AwDv6COiwlKmXu+/1rR42
Australia Central	ecdsa-sha2-nistp256	m2Hct3EsVmlVBMuwo9jsQd9hJzF	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAibmlzdHAyNTYAAABBE1XRuNjbnDPWF84vN
Australia Central	ecdsa-sha2-nistp384	uoYLwsgrkLp4D5diAulDKlb7C5nT	AAAAE2VjZHNhLXNoYTItbmlzdHAz0DQAAAibmlzdHAz0DQAAABhBARO/V15TyirrsNZZki
North Central US	rsa-sha2-256	9AV5CnZNkf9nd6W06WGNu7x6c4fd	AAAAB3NzaC1yc2EAAADAQABAAQDjt+aoDs1ngYi50PrR1lR6hz+k04D3hS0pgPTAj
North Central US	rsa-sha2-512	R3H1Mn2cnNb1X4qnHxdReba31GMF	AAAAB3NzaC1yc2EAAADAQABAAQDeM6M0S9Av7a5PghYLyM0T09ETbvdT9jgNe1rFn
North Central US	ecdsa-sha2-nistp256	6xMRs7dmIdi3vu0gn0f6x0Tb9R	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAibmlzdHAyNTYAAABBJw1dXty1YqVLjhAo1t
North Central US	ecdsa-sha2-nistp384	0cJKHHeTNQp17ewPTZwug5+/hfet	AAAAE2VjZHNhLXNoYTItbmlzdHAz0DQAAAibmlzdHAz0DQAAABhBGaqja46A9Q5PmhPzh
Brazil South	rsa-sha2-256	qNzx1kid41tZGcmbyZrz1CIPj9	AAAAB3NzaC1yc2EAAADAQABAAQ04g5K8emsS4Npl6jCT3wlp16Msb5ax6Qglef03IK
Brazil South	rsa-sha2-512	KAmGT8A7nRdxQD7gu1gmGTJvRhR	AAAAB3NzaC1yc2EAAADAQABAAQ6W0fiaS21Dze65r6CTB8rlBu1T1Zej+11m7Kt283
Brazil South	ecdsa-sha2-nistp256	rb0dmodk5Aq+nxHt04Tn7g6wyuw;	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAibmlzdHAyNTYAAABBNFqueXBigoFrM5Kp2e
Brazil South	ecdsa-sha2-nistp384	cenQeg58JZ+dvu3AC7P71C/Jq7v3	AAAAE2VjZHNhLXNoYTItbmlzdHAz0DQAAAibmlzdHAz0DQAAABhBhfn1fxV9/m6ZgOoq
UK West	rsa-sha2-256	2NQ5z6fQjt45ZKdViPS+I2kX7GoX	AAAAB3NzaC1yc2EAAADAQABAAQDNq0xtA0tdzmkSDTNgA05YLH5zuLFKd7RbrzuL4K
UK West	rsa-sha2-512	Mrfr1QmVjuk1Q5KvQ6YDyU1C3Ew	AAAAB3NzaC1yc2EAAADAQABAAQCIzODHJMnlu29q0dk1iwF05a0whbwIVu1JJFLg0
UK West	ecdsa-sha2-nistp256	bNYdYVgicv11yaOR/1xLqocxT8ba	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAibmlzdHAyNTYAAABBKwkoJuxB3E05bkjxnv
UK West	ecdsa-sha2-nistp384	6V8vLtrf615BjuLgToJ1cROM72Uq	AAAAE2VjZHNhLXNoYTItbmlzdHAz0DQAAAibmlzdHAz0DQAAABhBa+7R/5qSfsXAcMseiE
West Central US	rsa-sha2-256	aSNxepHr3CEijxbPB4D5I+v8um	AAAAB3NzaC1yc2EAAADAQABAAQDDNm08zD7dcfamYd/c1i2wYhRnaIgUmK7z/oehr
West Central US	rsa-sha2-512	vVHVYoH1ku1IZk+uZnStj3Qv2Ucy	AAAAB3NzaC1yc2EAAADAQABAAQ9Q8Tvvnea8hdq+t+Szr4XN1jEiR43nX6vrhcs6y
West Central US	ecdsa-sha2-nistp256	rKHjcTK2BvryQAFvjugTHdbpYBgF	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAibmlzdHAyNTYAAABBKmjEAUTitG+fSeoCM
West Central US	ecdsa-sha2-nistp384	gS9SYvaH6dCqyugArvFb13cw18q9	AAAAE2VjZHNhLXNoYTItbmlzdHAz0DQAAAibmlzdHAz0DQAAABhBD0HqM8ubcDRBmwuruX
Central US	rsa-sha2-256	G0Pn34T1cCkLH00xJLwmkEphxxKKk	AAAAB3NzaC1yc2EAAADAQABAAQ9oA4N2MxbHjcdSr01JodIpjTB2LPQUmwJj+a12K
Central US	rsa-sha2-512	VlhBzJhzoNRMyRs3Gyvk2rgacj	AAAAB3NzaC1yc2EAAADAQABAAQDPnuJixcZd6gAiifABQ377Mn0ZootRmdjs1J3R8/u
Central US	ecdsa-sha2-nistp256	qN1Fm+zCCQ4xEkNTarKiQduCd95+	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAibmlzdHAyNTYAAABBN6KpNy9XB1lV6jsqyR
Central US	ecdsa-sha2-nistp384	9no3/m09BEugyFxhaChilKiwyay	AAAAE2VjZHNhLXNoYTItbmlzdHAz0DQAAAibmlzdHAz0DQAAABhC1Eyrlw9pskKzDp/6H
North Europe	rsa-sha2-256	vTEosEjvg/jHYH1xIwf2rKrtEN1I	AAAAB3NzaC1yc2EAAADAQABAAQCanDNw172Rm12j6ZehRs4/twoeDE4HThgks5DRgRF
North Europe	rsa-sha2-512	c4FqTQY/IjTcovY/g7RRx0VSSo0b	AAAAB3NzaC1yc2EAAADAQABAAQCanDNw172Rm12j6ZehRs4/twoeDE4HThgks5DRgRF
North Europe	ecdsa-sha2-nistp256	wUF5N8vjGtNa/PYBVzQrhcrMgHuC	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAibmlzdHAyNTYAAABBC4oFTmr3wzccXcayC
North Europe	ecdsa-sha2-nistp384	w7dzF6D42eE2dfg/G1073dh+Qa2	AAAAE2VjZHNhLXNoYTItbmlzdHAz0DQAAAibmlzdHAz0DQAAABhBLgyasQj6FyeRa1jiQE
UAE North	rsa-sha2-256	Vazz+KIAdh85GQHAY1rl1tTY8/c	AAAAB3NzaC1yc2EAAADAQABAAQDRGQHLLR9ruI0GcNF2u3EpS2CbHdZ1qcgSR1bkaOX
UAE North	rsa-sha2-512	N0eTPUor20uTdgSjlhSaqj1TJU	AAAAB3NzaC1yc2EAAADAQABAAQDQAx9Lf1yVmwdG/rjQeHi1HTMwYaE/mMP6rxmfs9/I-
UAE North	ecdsa-sha2-nistp256	vAuGgsr0IQn0LuawCCOBt+jg0DV5	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAibmlzdHAyNTYAAABBEYpnxgANJN4IIvSw
UAE North	ecdsa-sha2-nistp384	A5fa4Pzkd10H2KVx1N1Eqk0hPzB	AAAAE2VjZHNhLXNoYTItbmlzdHAz0DQAAAibmlzdHAz0DQAAABhB0z4EndgFpo547D5XC
Germany Westcentral	rsa-sha2-256	0SKtGye+E9pp4QltwNLLiPSx+qKv	AAAAB3NzaC1yc2EAAADAQABAAQDsbkjxK9IJP8K98j+4/wGJQVdk0/x0MsF89wpjd/3
Germany Westcentral	rsa-sha2-512	90Y07Hn5p+JjeGGvsTsanhk3rm+	AAAAB3NzaC1yc2EAAADAQABAAQCrwSTqa0GD36iyMta4ZxSMz3mf51MIhK6APQ2snhR5
Germany Westcentral	ecdsa-sha2-nistp256	Ce+h+7thT5t75ypIkWZ6+jnM0Mz	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAibmlzdHAyNTYAAABBBmVE0InhtrK183oB4
Germany Westcentral	ecdsa-sha2-nistp384	hhQQi2irj5X5d9c+4714hAfvtA3c	AAAAE2VjZHNhLXNoYTItbmlzdHAz0DQAAAibmlzdHAz0DQAAABhB01FF3ceA17ZFERfvij
Switzerland West	rsa-sha2-256	yoVjbjB+U4Cp/ZpMgKku19t2pIF	AAAAB3NzaC1yc2EAAADAQABAAQDf19N03CjyKtdYxIgCjywIx1T1ppJ0m/ykv2zDz
Switzerland West	rsa-sha2-512	UgWxFaVY0YYMiNQ82Wt3D1Ldg3xt	AAAAB3NzaC1yc2EAAADAQABAAQ6svukqfg7147raZzrA1bZFO0/EDFgi+wRxs0ofH/
Switzerland West	ecdsa-sha2-nistp256	5Myz1uHqIMdh/+QEnbr3Zm6/HnsL	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAibmlzdHAyNTYAAABBEj5nxHejkV1lcfr9Ff
Switzerland West	ecdsa-sha2-nistp384	nS9RIUnm5ULmNIG+d7qSe11/kNzU	AAAAE2VjZHNhLXNoYTItbmlzdHAz0DQAAAibmlzdHAz0DQAAABhBB/Ps4Wp15xhNenavSH
Sweden Central	rsa-sha2-256	feu0rfEf3KhvHGFhxEjcuFcPt11+f	AAAAB3NzaC1yc2EAAADAQABAAQBo0imUzzHr0DxrjdWEpqkrBudLw2P2dvPE9doaXN

Region	Host Key Type	SHA 256 Fingerprint	Public Key
Sweden Central	rsa-sha2-512	5fx+Ic5p/MMR6TzvjJ2yrb4HMhw	AAAAB3NzaC1yc2EAAAQABAAQc2nRaxWtg4KGLC1TzLQ5qPZPyQ/YxbH4prjhg1uk
Sweden Central	ecdsa-sha2-nistp256	6HikgYBMSL9VguDq9bmwRaVxD0IL	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBErZhZNmDhMKbSUXLB
Sweden Central	ecdsa-sha2-nistp384	apRb96GLQ3LZ3E+r2dyr9imMDx	AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAIBmlzdHAzODQAAABhBKA5kwsqDkzZwmQCjIF
East Asia	rsa-sha2-256	XYUEB+zABdpXRkla8RCoWy4hZp9	AAAAB3NzaC1yc2EAAAQABAAQDNk1aGhR1HdomU5uGvkceJFQ0mhmnwxsHAUNoGUhi
East Asia	rsa-sha2-512	FUYhL0FaN8Zkj/M0/VJnm8jPL+2k	AAAAB3NzaC1yc2EAAAQABAAQDNk1aGhR1HdomU5uGvkceJFQ0mhmnwxsHAUNoGUhi
East Asia	ecdsa-sha2-nistp256	/iq1i88fRFHFBw4DBtZUX7GRbt5d	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBCvI7dc7w3K919GK2VH
East Asia	ecdsa-sha2-nistp384	KssXE1Wc60a0dS2CnySg0kbv9	AAAAB3NzaC1yc2EAAAQABAAQCD2UBC1KeTx8/tQIxVEBuypcu/5n3B/g0zqE7tFmPYi
South Africa North	rsa-sha2-256	qu1qry+E/fBbRtDoO+CdKiLxxKnF	AAAAB3NzaC1yc2EAAAQABAAQCD2UBC1KeTx8/tQIxVEBuypcu/5n3B/g0zqE7tFmPYi
South Africa North	rsa-sha2-512	1/ogzd+xjh3itFg3ipAyA2pwj1o3	AAAAB3NzaC1yc2EAAAQABAAQCD2UBC1KeTx8/tQIxVEBuypcu/5n3B/g0zqE7tFmPYi
South Africa North	ecdsa-sha2-nistp256	e6v7pRdZE0i1u2/VePcQLguy7d+5	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBEQemJxERZKre+A+MA
South Africa North	ecdsa-sha2-nistp384	NmxPlxzK2GpozWY374nvAFnYUBwJ	AAAAB3NzaC1yc2EAAAQABAAQDIwNEfrP6Httmm5GoxwprQ57AyD6b3E0Ve5pTQGWI
UK South	rsa-sha2-256	3nrDdwU0wG0XgrFrFgW27xhWSizt	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
UK South	rsa-sha2-512	Csn18SFb1kdppVsJc1jNVSync2eDw	AAAAB3NzaC1yc2EAAAQABAAQDIwNEfrP6Httmm5GoxwprQ57AyD6b3E0Ve5pTQGWI
UK South	ecdsa-sha2-nistp256	weMVz0mQn1MdMp5XB0u9SdN5meBt	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
UK South	ecdsa-sha2-nistp384	HpsZzo0CCsUbD3nA0txpuK1vnE	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
Australia Southeast	rsa-sha2-256	YafIMxu7NtoKCrjQ2eDxoRKHbZ	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
Australia Southeast	rsa-sha2-512	FpFC0sNUkdnd1k0SkwD1fnasYhE	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
Australia Southeast	ecdsa-sha2-nistp256	4xc49pnNg4t/tr91pdtbZLdkqzQv	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
Australia Southeast	ecdsa-sha2-nistp384	DEyjMyyAYkegwLtmBROR/8StR1kN	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
France South	rsa-sha2-256	aywTR4YJBQrwWsIAlxC1lDDHpj3	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
France South	rsa-sha2-512	+y5oZsLMVG6kf1H1tp475WoKuqh	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
France South	ecdsa-sha2-nistp256	LHW1PtDIQAH1MkOagvMJU00wR41	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
France South	ecdsa-sha2-nistp384	btqtCD/hJwVahHwz/qftHV3B+ezJ	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
West US 2	rsa-sha2-256	ktnBebdoHk7eDo2tvJuv36XnMtfa	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
West US 2	rsa-sha2-512	i8v3Xxh/phaa5EyEr5NM4nTSC/R	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
West US 2	ecdsa-sha2-nistp256	rt5kaA0neIFDIWTP2MjWk9c0Sapz	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
West US 2	ecdsa-sha2-nistp384	g0vDKd4G5MknxWewbnCYahCv11zg	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
South India	rsa-sha2-256	SgFLJvQvQodZxKb13DnGywfp9d1i	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
South India	rsa-sha2-512	T4mrHCEhbFNQ5Sng/_m0ViU/hxF1	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
South India	ecdsa-sha2-nistp256	7PQhzR5S6sEFYkn2s3GxK6k8bwHg	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
South India	ecdsa-sha2-nistp384	sXR2nhTTnof58neK+Xjm9pu8miE	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
Japan West	rsa-sha2-256	DRVSSje7BbY1VJcfXqLzIznCbVU4	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
Japan West	rsa-sha2-512	yL19t2j1krTVwAxsZ59wbpq+ZCnw	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
Japan West	ecdsa-sha2-nistp256	VYwgC6A4o1B7MoliEkuF2zhgdz	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
Japan West	ecdsa-sha2-nistp384	+gvZr0QRq31VOuqdqgsawkvj6v/	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
Norway East	rsa-sha2-256	vmcel/XoNut7ysR079fp5wAKyhT	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
Norway East	rsa-sha2-512	JZPRhXmx44tEnxp+wPvexDys1tSy	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
Norway East	ecdsa-sha2-nistp256	mE43kdMTv2ioQxwHD7z+vV4	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
Norway East	ecdsa-sha2-nistp384	cKF2asIQu0Uv0C/wau4ex9ioVt	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi
France Central	rsa-sha2-256	zYLhY1rtM2sgP5vwyCtaU8v2is1d	AAAAB3NzaC1yc2EAAAQABAAQCDLm+90R0p5zrc6nLKBjWnrTnUeCe0n1v9Y3qWi

Region	Host Key Type	SHA 256 Fingerprint	Public Key
France Central	rsa-sha2-512	ixum/Dragma5DAMBza/c5/MY02fj AAAAB3NzaC1yc2EAAAQABAAQDjtJ9EvMFwiBCCmYF0z02GaWZJXLc7F5QrvFv6Nm	
France Central	ecdsa-sha2-nistp256	N61PH8SVCAx0q7z7eIV4mRnotafm AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBBK3UBFa/Ke9y3aLs1q1	
France Central	ecdsa-sha2-nistp384	/CkQnHAs57ehNeC9ZhTyvVr8yVj AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAIBmlzdHAzODQAAABhBG/x6qX+DrtmxOoMzwe	
West US 3	rsa-sha2-256	pOKzaf3mrTJhfdR/9dbodyNza30t AAAAB3NzaC1yc2EAAAQABAAQDKEfBaFSLsI28jdc854Rq6AL9Ku8g8L+0WQfwvb1	
West US 3	rsa-sha2-512	KKcoWCeuJeepexnJCxoFqKJM8Xr AAAAB3NzaC1yc2EAAAQABAAQDQhVgDjCIarGeJKgmSxRh4wVjW6PxKbNk3d0M4	
West US 3	ecdsa-sha2-nistp256	j4N1ZP/wOKNnM3MKNEcTksqwBdF0 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBBETvvRvehA02010FFt	
West US 3	ecdsa-sha2-nistp384	DkJet/6Pm6EXfpz2U6aaHJ940vj AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAIBmlzdHAzODQAAABhEu+HpgDp@02m11jD	
Central India	rsa-sha2-256	OcX6wPaofbb+UG/1LYr30CPhtnKL AAAAB3NzaC1yc2EAAAQABAAQDwUkb0JR8ZzqhE3k2HMBW099rNwoHwJa+PVWPQHyC	
Central India	rsa-sha2-512	HSGc5u8s+QILdyBq6wGJkxRck5nx AAAAB3NzaC1yc2EAAAQABAAQDSO/R/yw8q3yLkSHow0Bi2wKDwQPr118sk3hdRU	
Central India	ecdsa-sha2-nistp256	zBKgtf770MPVvxgql1/4pJinGPjC AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBBjHx8+PF0VBsp1619X	
Central India	ecdsa-sha2-nistp384	PzKXw0/DR/KnUElcvWIwSdabp62 AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAIBmlzdHAzODQAAABhBjwEy1f+GYN4rxh1CAk	
Korea South	rsa-sha2-256	J1w5chMr9yRceU2fpqyvhwEQLG7j AAAAB3NzaC1yc2EAAAQABAAQDHxQIUGNU0L+st+FkEsghv+Hxq4k7EZIKMT9102v	
Korea South	rsa-sha2-512	sHzKpDvhndbXaRaFJUskmpCCB3Hg AAAAB3NzaC1yc2EAAAQABAAQCFGUUmJiogHgbhxjEunkoALMjG77m+jgZquj03MwTi	
Korea South	ecdsa-sha2-nistp256	XMSxNHAwcySCSwxEMUMIFCoNJU/g AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBBHTp085Vgs16/SEJwg	
Korea South	ecdsa-sha2-nistp384	6T8uMI9gcG3HtjYuyqNNxi99kgsh AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAIBmlzdHAzODQAAABhAgPPIDWZqvB/kuIguF	
South Central US	rsa-sha2-256	n7P8Nrxy8pWNSaNIh8t5Xi9rXii AAAAB3NzaC1yc2EAAAQABAAQD4Pg88PxPPpGfvrlUGSiidFifkrk2/u10mhcoGoIf	
South Central US	rsa-sha2-512	B2o0tHpxZwezb1rKxGNC3QJLQ6 AAAAB3NzaC1yc2EAAAQABAAQAC+LJA8w3BcwITzJv6CAkx/0HPBdy3LjKPK2NQgV9m	
South Central US	ecdsa-sha2-nistp256	Wg9hT1PmrRH9aC91TSf8hGqa85A AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBBjEz4iwyq7aaBNK1Ab	
South Central US	ecdsa-sha2-nistp384	rgRhPelmxAix6TBDAhmGQxnkjDIm AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAIBmlzdHAzODQAAABhBxKXGbWfVe18G9gbCxF	
Korea Central	rsa-sha2-256	Eky+yOmufAfsZtf4w7ToRcw0evgZ AAAAB3NzaC1yc2EAAAQABAAQDyCytAe7QtAd3lmH+41KJNEBNWnPUB+PELE9f4us5G	
Korea Central	rsa-sha2-512	KAji7Q8E21T3+1se7h74L6rfPnLe AAAAB3NzaC1yc2EAAAQABAAQDxZYb5eIwhmSwNu6G9FFDRgq1zjYorMSXJ4swH	
Korea Central	ecdsa-sha2-nistp256	XjVSEyG1Btk0Ndvdw11tA0X1nPw AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBPYiomalB3RoXzvdFqi	
Korea Central	ecdsa-sha2-nistp384	p/jQTk9VsbsKZyK09opVQVi3HzvJ AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAIBmlzdHAzODQAAABhBN3NA7u4ZC57ibkCA	
Southeast Asia	rsa-sha2-256	f0cyRMVxUgtpsa9J6pwAMsk2MY/ AAAAB3NzaC1yc2EAAAQABAAQDwP6PAGMTdzNkwKzt+A3dhbnete6jyLboOXwdv/	
Southeast Asia	rsa-sha2-512	vh8Uh40NCD31HVh5KEcURUzrT3hi AAAAB3NzaC1yc2EAAAQABAAQCDl+E/W2Rpjm1wMrg5EtMs0AE7BF2q5jnxxaIBwq	
Southeast Asia	ecdsa-sha2-nistp256	q70sE02p9S26E63b+Mxr1wb15wf AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBBEvbjkwSA0RQuT2nQf8.	
Southeast Asia	ecdsa-sha2-nistp384	HpnueSwbRGeiqGEAKsXF0Htjvc AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAIBmlzdHAzODQAAABhBMGAMUN+oyuXuf6rkS	
Australia East	rsa-sha2-256	MrPZLU81lsG+SzgBN8eH702H4zuu AAAAB3NzaC1yc2EAAAQABAAQDsRwH+DKINZZNP0G0617mFIgTrnJy7A/1Vb9/2/	
Australia East	rsa-sha2-512	jKDaVBMh+d9CUjq8tQH5LnC1pc9 AAAAB3NzaC1yc2EAAAQABAAQDFFirQKaYkcecqdutyM0r1efwiaIM/h302KjR0io	
Australia East	ecdsa-sha2-nistp256	s8NdoxI0mdWchKMmt/oYtn1FNAD8 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBBKG2nz5SnoR5KVYAnB	
Australia East	ecdsa-sha2-nistp384	YmeF1kX0R0W/ssqzKCKjLoSL3Cc1 AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAIBmlzdHAzODQAAABhBFj5neNPCIxkYs7HK	
Japan East	rsa-sha2-256	P3w0FZQMpmpRFBtnIQH2R88eWc+f AAAAB3NzaC1yc2EAAAQABAAQDzRgA9jAgcr1g1Zro0+IVzkLDCo79	
Japan East	rsa-sha2-512	4adNtgbPGYD+r/yLQzfSpkrI9z AAAAB3NzaC1yc2EAAAQABAAQDzRgA9jAgcr1g1Zro0+IVzkLDCo79	
Japan East	ecdsa-sha2-nistp256	Ift/j4bH2Jc0UhuUAdfcy3Tvesc AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBBKVq+uiJxm11ys367ir	
Japan East	ecdsa-sha2-nistp384	9XLsgx1qDtoZosvWz/m7418Hwdc AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAIBmlzdHAzODQAAABhFh7i1cfuoXeyAgxs+l	
Canada East	rsa-sha2-256	SRhd9gnvJS630A8VtCYMqc4djz5R AAAAB3NzaC1yc2EAAAQABAAQD2nSByIh/Nc3Zhsjk3zt7mspcUUXcq9Y/jc9Qqsfh	
Canada East	rsa-sha2-512	60yzcSS0H1ubdGkuNPWMXB9j21Hq AAAAB3NzaC1yc2EAAAQABAAQDm4meGZwdDzrgA9jAgcr1g1Zro0+IVzkLDCo79	
Canada East	ecdsa-sha2-nistp256	YPqDobCavDz/GV7FuR/gzYqgUzI AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBBK1fnJ9/r105/YgeGle	
Canada East	ecdsa-sha2-nistp384	Y6FK9rwscBkyKN7mgPAEj0jkFxrv AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAIBmlzdHAzODQAAABhBDS8gYaqmJ8eEjmDF2	
Canada Central	rsa-sha2-256	K0YkeGvx4egh9DTgx10NDMvS1ke AAAAB3NzaC1yc2EAAAQABAAQC7jhZvp5GMryA2gyjbQXTc/QoSeDe1uBupj16ndy	

Region	Host Key Type	SHA 256 Fingerprint	Public Key
Canada Central	rsa-sha2-512	tdixmLr++BvpFMpiWyVkr5iAXM4t	AAAAB3NzaC1yc2EAAAQABAAQDNMzWl0AuF2Uyn4NIK+XdaHuon2jEBwUFNSAx04JP
Canada Central	ecdsa-sha2-nistp256	HhbplbdxrinWvNsk/OvkowI9nw	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBBuyYEUpBjzEnYljSwk
Canada Central	ecdsa-sha2-nistp384	EjeadkKaEgaNfdwTz1qanUbDigz	AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAIBmlzdHAzODQAAABhBORAcpaBXkmSuylClbAO
Switzerland North	rsa-sha2-256	4cxg5pca9HcvAxDMrE7GdwvUz15R	AAAAB3NzaC1yc2EAAAQABAAQcQqSS6hVsmykLqNCqZnt0ao0QSS1xG89BiwNaR7uQ
Switzerland North	rsa-sha2-512	E631mwP5d5aK3wJLj4ksx0wPab1	AAAAB3NzaC1yc2EAAAQABAAQcQqSS6hVsmykLqNCqZnt0ao0QSS1xG89BiwNaR7uQ
Switzerland North	ecdsa-sha2-nistp256	DfyPsw04f2rU6Pxelx8iVRu+hrtS	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBBJICveabT6GFbyaCSe
Switzerland North	ecdsa-sha2-nistp384	Rw0TLDVU4PqsXb0unR2BZcn2/wqF	AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAIBmlzdHAzODQAAABhBLhGaEyHyfvVU051mK
UAE Central	rsa-sha2-256	GW5lrSx75BsJFe4y4vwJFdg454fr	AAAAB3NzaC1yc2EAAAQABAAQDAQEpj9zkZ8F3idkDdbZv4A3+1RC/0Un6HZVvv5M
UAE Central	rsa-sha2-512	zf1L4o1l2bg9JcPA/qFvT2jSYw	AAAAB3NzaC1yc2EAAAQABAAQDAtxSg71HzGfc1wVuErZZo6VG5uaWly1khb67rJS
UAE Central	ecdsa-sha2-nistp256	P3KxgoZgjHHxid66gbkRETjPsHus	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBBovHAXCWC9HGJn5SRW
UAE Central	ecdsa-sha2-nistp384	E+jKxd6hnFVIXPQYreABxpZB7tpf	AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAIBmlzdHAzODQAAABhBMDLyroqceuIpmdQk/g
Germany North	rsa-sha2-256	ppHn1ruDLR73KzW/m7yc3dHQ0Jvx	AAAAB3NzaC1yc2EAAAQABAAQDNNjCcDxjl3ess1Q0Ekbn5bPyxXpedk32ZX4oTc
Germany North	rsa-sha2-512	m/oFTRHkc3HxfhCKk1+jY1rPJrT9	AAAAB3NzaC1yc2EAAAQABAAQDkN3CN1V1TaHy/CduQaZIKuKSC/+oX19sYntRdgCb
Germany North	ecdsa-sha2-nistp256	F4o8Z911B5SrP0faYFwKMQtNw/+J	AAAAB3NzaC1yc2EAAAQABAAQDnVz5RvnGa0F5n+Nx3nt526fmF+Vuf0Yo5/
Germany North	ecdsa-sha2-nistp384	BgW5e9lciYG1oIxoinVnUcpdh3Jpn	AAAAB3NzaC1yc2EAAAQABAAQDnVz5RvnGa0F5n+Nx3nt526fmF+Vuf0Yo5/
Australia Central 2	rsa-sha2-256	sqvq1zdD30iAbnDjs0/why2c3l	AAAAB3NzaC1yc2EAAAQABAAQDnVz5RvnGa0F5n+Nx3nt526fmF+Vuf0Yo5/
Australia Central 2	rsa-sha2-512	p6vLHCTBcK0kqz7eiVCT6pLuIg7h	AAAAB3NzaC1yc2EAAAQABAAQDcQd2zICW1RLkweRMG9wt0GxA5unQ/0nd9ys1f0I
Australia Central 2	ecdsa-sha2-nistp256	m7Go9P1bwPcPHACZzRSXdwYr0D1dz	AAAAB3NzaC1yc2EAAAQABAAQDnVz5RvnGa0F5n+Nx3nt526fmF+Vuf0Yo5/
Australia Central 2	ecdsa-sha2-nistp384	9Jc390ueTg3pQcq8KJgzsmPlVxxI	AAAAB3NzaC1yc2EAAAQABAAQDnVz5RvnGa0F5n+Nx3nt526fmF+Vuf0Yo5/
South Africa West	rsa-sha2-256	aMMzaNmXR+V1NrwlMovyKwfBkQ6	AAAAB3NzaC1yc2EAAAQABAAQDGe98UTn1jsYaeJwtp3ABvT/hZP6Mp1r5beyJ2Sw
South Africa West	rsa-sha2-512	Uc7QB0FT4NgypB34GCat8G4j1ZBx	AAAAB3NzaC1yc2EAAAQABAAQDGe98UTn1jsYaeJwtp3ABvT/hZP6Mp1r5beyJ2Sw
South Africa West	ecdsa-sha2-nistp256	pr1KB8apI+FNLKkzvUxX0/waiqb	AAAAB3NzaC1yc2EAAAQABAAQDGe98UTn1jsYaeJwtp3ABvT/hZP6Mp1r5beyJ2Sw
South Africa West	ecdsa-sha2-nistp384	A3RFM0d6GgU1crkXL1YRKNXIdA8	AAAAB3NzaC1yc2EAAAQABAAQDGe98UTn1jsYaeJwtp3ABvT/hZP6Mp1r5beyJ2Sw
Jio India West	rsa-sha2-256	hcy1XbIniEZloraGrvecJcvlwzZ	AAAAB3NzaC1yc2EAAAQABAAQDGe98UTn1jsYaeJwtp3ABvT/hZP6Mp1r5beyJ2Sw
Jio India West	rsa-sha2-512	LPctDLIz/vqg4POMOpq1yD9EE9s	AAAAB3NzaC1yc2EAAAQABAAQDGe98UTn1jsYaeJwtp3ABvT/hZP6Mp1r5beyJ2Sw
Jio India West	ecdsa-sha2-nistp256	mBx6CZ+6DseVrwsKfkNph9NdrVlg	AAAAB3NzaC1yc2EAAAQABAAQDGe98UTn1jsYaeJwtp3ABvT/hZP6Mp1r5beyJ2Sw
Jio India West	ecdsa-sha2-nistp384	lWqX9Yfn7uDz/8gXpG4sZcWlCAox	AAAAB3NzaC1yc2EAAAQABAAQDGe98UTn1jsYaeJwtp3ABvT/hZP6Mp1r5beyJ2Sw
Sweden South	rsa-sha2-256	kS1NUherycqJAYe8KZi8AnqIQ9ud	AAAAB3NzaC1yc2EAAAQABAAQDGe98UTn1jsYaeJwtp3ABvT/hZP6Mp1r5beyJ2Sw
Sweden South	rsa-sha2-512	G+oX014UJXR0t1xHrC1715XuohBk	AAAAB3NzaC1yc2EAAAQABAAQDGe98UTn1jsYaeJwtp3ABvT/hZP6Mp1r5beyJ2Sw
Sweden South	ecdsa-sha2-nistp256	8C148yiGdrJCGF6HpdzINhgk5AA	AAAAB3NzaC1yc2EAAAQABAAQDGe98UTn1jsYaeJwtp3ABvT/hZP6Mp1r5beyJ2Sw
Sweden South	ecdsa-sha2-nistp384	ra8+vb8aSkTbs00KAxDr121n9p41	AAAAB3NzaC1yc2EAAAQABAAQDGe98UTn1jsYaeJwtp3ABvT/hZP6Mp1r5beyJ2Sw
Jio India Central	rsa-sha2-256	DmNCjG1VjXwWmrXw5USD0pAnJAbE	AAAAB3NzaC1yc2EAAAQABAAQDGe98UTn1jsYaeJwtp3ABvT/hZP6Mp1r5beyJ2Sw
Jio India Central	rsa-sha2-512	m2P7vnys12adTz/0P6ebSR7Xx8Au	AAAAB3NzaC1yc2EAAAQABAAQDGe98UTn1jsYaeJwtp3ABvT/hZP6Mp1r5beyJ2Sw
Jio India Central	ecdsa-sha2-nistp256	zA0A1pk0Xz8Vr/Def8ztPaLcivx	AAAAB3NzaC1yc2EAAAQABAAQDGe98UTn1jsYaeJwtp3ABvT/hZP6Mp1r5beyJ2Sw
Jio India Central	ecdsa-sha2-nistp384	OTG7jxUSj+xRdl28JpYAh5fr6tfc	AAAAB3NzaC1yc2EAAAQABAAQDGe98UTn1jsYaeJwtp3ABvT/hZP6Mp1r5beyJ2Sw
Brazil Southeast	rsa-sha2-256	D+S7uHDWly0VPrRg9mlak70PBpghB	AAAAB3NzaC1yc2EAAAQABAAQDGe98UTn1jsYaeJwtp3ABvT/hZP6Mp1r5beyJ2Sw
Brazil Southeast	rsa-sha2-512	C+p2eAPf5uec0Yg+aeoVAAloAf8	AAAAB3NzaC1yc2EAAAQABAAQDGe98UTn1jsYaeJwtp3ABvT/hZP6Mp1r5beyJ2Sw
Brazil Southeast	ecdsa-sha2-nistp256	dhADLmzQOE0mPcc7S3wV+x2AU1v1	AAAAB3NzaC1yc2EAAAQABAAQDGe98UTn1jsYaeJwtp3ABvT/hZP6Mp1r5beyJ2Sw
Brazil Southeast	ecdsa-sha2-nistp384	mjFHEltgAJkPTw04dc7pzVvnJ6wl	AAAAB3NzaC1yc2EAAAQABAAQDGe98UTn1jsYaeJwtp3ABvT/hZP6Mp1r5beyJ2Sw
Norway West	rsa-sha2-256	Ea3Vj3EfZYm25Ax1Aty30Ad+1hx	AAAAB3NzaC1yc2EAAAQABAAQDGe98UTn1jsYaeJwtp3ABvT/hZP6Mp1r5beyJ2Sw

REGION	HOST KEY TYPE	SHA 256 FINGERPRINT	PUBLIC KEY
Norway West	rsa-sha2-512	uHGFIB97I8y8nSAEc1D7InBKzAxS	AAAAB3NzaC1yc2EAAAQABAAQDPXLVCb1kqh8gERY43bvyPcfxV0UnZyWshKEK5+QT
Norway West	ecdsa-sha2-nistp256	mujUcRHpId06YvSLxboTHWmq0pL	AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAIBmlzdHAyNTYAAABBoefohG21zu2JGcUvjk
Norway West	ecdsa-sha2-nistp384	Q1zJV54Ggw1AObzQjGt/J2TQ1kT	AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAIBmlzdHAzODQAAABhBNYnNgJKaYCBYLPdh21

<sup>1</sup> The SHA 256 fingerprint is used by Open SSH and WinSCP.

## See also

- [SSH File Transfer Protocol \(SFTP\) support for Azure Blob Storage](#)
- [Connect to Azure Blob Storage by using the SSH File Transfer Protocol \(SFTP\)](#)
- [Limitations and known issues with SSH File Transfer Protocol \(SFTP\) support for Azure Blob Storage](#)
- [SSH File Transfer Protocol \(SFTP\) performance considerations in Azure Blob storage](#)

# Azure Policy built-in definitions for Azure Storage

8/22/2022 • 12 minutes to read • [Edit Online](#)

This page is an index of [Azure Policy](#) built-in policy definitions for Azure Storage. For additional Azure Policy built-ins for other services, see [Azure Policy built-in definitions](#).

The name of each built-in policy definition links to the policy definition in the Azure portal. Use the link in the **Version** column to view the source on the [Azure Policy GitHub repo](#).

## Microsoft.Storage

NAME (AZURE PORTAL)	DESCRIPTION	EFFECT(S)	VERSION (GITHUB)
[Preview]: [Preview]: Configure backup for blobs on storage accounts with a given tag to an existing backup vault in the same region	Enforce backup for blobs on all storage accounts that contain a given tag to a central backup vault. Doing this can help you manage backup of blobs contained across multiple storage accounts at scale. For more details, refer to <a href="https://aka.ms/AB-BlobBackupAzPolicies">https://aka.ms/AB-BlobBackupAzPolicies</a>	DeployIfNotExists, AuditIfNotExists, Disabled	<a href="#">2.0.0-preview</a>
[Preview]: [Preview]: Configure blob backup for all storage accounts that do not contain a given tag to a backup vault in the same region	Enforce backup for blobs on all storage accounts that do not contain a given tag to a central backup vault. Doing this can help you manage backup of blobs contained across multiple storage accounts at scale. For more details, refer to <a href="https://aka.ms/AB-BlobBackupAzPolicies">https://aka.ms/AB-BlobBackupAzPolicies</a>	DeployIfNotExists, AuditIfNotExists, Disabled	<a href="#">2.0.0-preview</a>
[Preview]: [Preview]: Storage account public access should be disallowed	Anonymous public read access to containers and blobs in Azure Storage is a convenient way to share data but might present security risks. To prevent data breaches caused by undesired anonymous access, Microsoft recommends preventing public access to a storage account unless your scenario requires it.	audit, Audit, deny, Deny, disabled, Disabled	<a href="#">3.1.0-preview</a>

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Azure File Sync should use private link</a>	Creating a private endpoint for the indicated Storage Sync Service resource allows you to address your Storage Sync Service resource from within the private IP address space of your organization's network, rather than through the internet-accessible public endpoint. Creating a private endpoint by itself does not disable the public endpoint.	AuditIfNotExists, Disabled	1.0.0
<a href="#">Configure Azure File Sync with private endpoints</a>	A private endpoint is deployed for the indicated Storage Sync Service resource. This enables you to address your Storage Sync Service resource from within the private IP address space of your organization's network, rather than through the internet-accessible public endpoint. The existence of one or more private endpoints by themselves does not disable the public endpoint.	DeployIfNotExists, Disabled	1.0.0
<a href="#">Configure diagnostic settings for storage accounts to Log Analytics workspace</a>	Deploys the diagnostic settings for storage accounts to stream resource logs to a Log Analytics workspace when any storage account which is missing this diagnostic settings is created or updated.	DeployIfNotExists, Disabled	1.3.0
<a href="#">Configure Storage account to use a private link connection</a>	Private endpoints connect your virtual network to Azure services without a public IP address at the source or destination. By mapping private endpoints to your storage account, you can reduce data leakage risks. Learn more about private links at - <a href="https://aka.ms/azureprivatelinkoverview">https://aka.ms/azureprivatelinkoverview</a>	DeployIfNotExists, Disabled	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Configure storage accounts to disable public network access</a>	<p>To improve the security of Storage Accounts, ensure that they aren't exposed to the public internet and can only be accessed from a private endpoint. Disable the public network access property as described in <a href="https://aka.ms/storageaccountspublicnetworkaccess">https://aka.ms/storageaccountspublicnetworkaccess</a>.</p> <p>This option disables access from any public address space outside the Azure IP range, and denies all logins that match IP or virtual network-based firewall rules. This reduces data leakage risks.</p>	Modify, Disabled	1.0.1
<a href="#">Deploy Advanced Threat Protection on storage accounts</a>	This policy enables Advanced Threat Protection on storage accounts.	DeployIfNotExists, Disabled	1.0.0
<a href="#">Geo-redundant storage should be enabled for Storage Accounts</a>	Use geo-redundancy to create highly available applications	Audit, Disabled	1.0.0
<a href="#">HPC Cache accounts should use customer-managed key for encryption</a>	Manage encryption at rest of Azure HPC Cache with customer-managed keys. By default, customer data is encrypted with service-managed keys, but customer-managed keys are commonly required to meet regulatory compliance standards. Customer-managed keys enable the data to be encrypted with an Azure Key Vault key created and owned by you. You have full control and responsibility for the key lifecycle, including rotation and management.	Audit, Disabled, Deny	2.0.0
<a href="#">Modify - Configure Azure File Sync to disable public network access</a>	The Azure File Sync's internet-accessible public endpoint are disabled by your organizational policy. You may still access the Storage Sync Service via its private endpoint(s).	Modify, Disabled	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Public network access should be disabled for Azure File Sync</a>	<p>Disabling the public endpoint allows you to restrict access to your Storage Sync Service resource to requests destined to approved private endpoints on your organization's network.</p> <p>There is nothing inherently insecure about allowing requests to the public endpoint, however, you may wish to disable it to meet regulatory, legal, or organizational policy requirements. You can disable the public endpoint for a Storage Sync Service by setting the incomingTrafficPolicy of the resource to AllowVirtualNetworksOnly.</p>	Audit, Deny, Disabled	1.0.0
<a href="#">Queue Storage should use customer-managed key for encryption</a>	<p>Secure your queue storage with greater flexibility using customer-managed keys. When you specify a customer-managed key, that key is used to protect and control access to the key that encrypts your data. Using customer-managed keys provides additional capabilities to control rotation of the key encryption key or cryptographically erase data.</p>	Audit, Deny, Disabled	1.0.0
<a href="#">Secure transfer to storage accounts should be enabled</a>	<p>Audit requirement of Secure transfer in your storage account. Secure transfer is an option that forces your storage account to accept requests only from secure connections (HTTPS). Use of HTTPS ensures authentication between the server and the service and protects data in transit from network layer attacks such as man-in-the-middle, eavesdropping, and session-hijacking</p>	Audit, Deny, Disabled	2.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Storage account containing the container with activity logs must be encrypted with BYOK</a>	This policy audits if the Storage account containing the container with activity logs is encrypted with BYOK. The policy works only if the storage account lies on the same subscription as activity logs by design. More information on Azure Storage encryption at rest can be found here <a href="https://aka.ms/azurestorage-byok">https://aka.ms/azurestorage-byok</a> .	AuditIfNotExists, Disabled	1.0.0
<a href="#">Storage account encryption scopes should use customer-managed keys to encrypt data at rest</a>	Use customer-managed keys to manage the encryption at rest of your storage account encryption scopes. Customer-managed keys enable the data to be encrypted with an Azure key-vault key created and owned by you. You have full control and responsibility for the key lifecycle, including rotation and management. Learn more about storage account encryption scopes at <a href="https://aka.ms/encryption-scopes-overview">https://aka.ms/encryption-scopes-overview</a> .	Audit, Deny, Disabled	1.0.0
<a href="#">Storage account encryption scopes should use double encryption for data at rest</a>	Enable infrastructure encryption for encryption at rest of your storage account encryption scopes for added security. Infrastructure encryption ensures that your data is encrypted twice.	Audit, Deny, Disabled	1.0.0
<a href="#">Storage account keys should not be expired</a>	Ensure the user storage account keys are not expired when key expiration policy is set, for improving security of account keys by taking action when the keys are expired.	Audit, Deny, Disabled	3.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Storage accounts should allow access from trusted Microsoft services</a>	Some Microsoft services that interact with storage accounts operate from networks that can't be granted access through network rules. To help this type of service work as intended, allow the set of trusted Microsoft services to bypass the network rules. These services will then use strong authentication to access the storage account.	Audit, Deny, Disabled	1.0.0
<a href="#">Storage accounts should be limited by allowed SKUs</a>	Restrict the set of storage account SKUs that your organization can deploy.	Audit, Deny, Disabled	1.1.0
<a href="#">Storage accounts should be migrated to new Azure Resource Manager resources</a>	Use new Azure Resource Manager for your storage accounts to provide security enhancements such as: stronger access control (RBAC), better auditing, Azure Resource Manager based deployment and governance, access to managed identities, access to key vault for secrets, Azure AD-based authentication and support for tags and resource groups for easier security management	Audit, Deny, Disabled	1.0.0
<a href="#">Storage accounts should disable public network access</a>	To improve the security of Storage Accounts, ensure that they aren't exposed to the public internet and can only be accessed from a private endpoint. Disable the public network access property as described in <a href="https://aka.ms/storageaccountpublicnetworkaccess">https://aka.ms/storageaccountpublicnetworkaccess</a> . This option disables access from any public address space outside the Azure IP range, and denies all logins that match IP or virtual network-based firewall rules. This reduces data leakage risks.	Audit, Deny, Disabled	1.0.1

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Storage accounts should have infrastructure encryption</a>	Enable infrastructure encryption for higher level of assurance that the data is secure. When infrastructure encryption is enabled, data in a storage account is encrypted twice.	Audit, Deny, Disabled	1.0.0
<a href="#">Storage accounts should have shared access signature (SAS) policies configured</a>	Ensure storage accounts have shared access signature (SAS) expiration policy enabled. Users use a SAS to delegate access to resources in Azure Storage account. And SAS expiration policy recommend upper expiration limit when a user creates a SAS token.	Audit, Deny, Disabled	1.0.0
<a href="#">Storage accounts should have the specified minimum TLS version</a>	Configure a minimum TLS version for secure communication between the client application and the storage account. To minimize security risk, the recommended minimum TLS version is the latest released version, which is currently TLS 1.2.	Audit, Deny, Disabled	1.0.0
<a href="#">Storage accounts should prevent cross tenant object replication</a>	Audit restriction of object replication for your storage account. By default, users can configure object replication with a source storage account in one Azure AD tenant and a destination account in a different tenant. It is a security concern because customer's data can be replicated to a storage account that is owned by the customer. By setting allowCrossTenantReplication to false, objects replication can be configured only if both source and destination accounts are in the same Azure AD tenant.	Audit, Deny, Disabled	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Storage accounts should prevent shared key access</a>	Audit requirement of Azure Active Directory (Azure AD) to authorize requests for your storage account. By default, requests can be authorized with either Azure Active Directory credentials, or by using the account access key for Shared Key authorization. Of these two types of authorization, Azure AD provides superior security and ease of use over Shared Key, and is recommended by Microsoft.	Audit, Deny, Disabled	1.0.0
<a href="#">Storage accounts should restrict network access</a>	Network access to storage accounts should be restricted. Configure network rules so only applications from allowed networks can access the storage account. To allow connections from specific internet or on-premises clients, access can be granted to traffic from specific Azure virtual networks or to public internet IP address ranges	Audit, Deny, Disabled	1.1.1
<a href="#">Storage accounts should restrict network access using virtual network rules</a>	Protect your storage accounts from potential threats using virtual network rules as a preferred method instead of IP-based filtering. Disabling IP-based filtering prevents public IPs from accessing your storage accounts.	Audit, Deny, Disabled	1.0.1
<a href="#">Storage Accounts should use a virtual network service endpoint</a>	This policy audits any Storage Account not configured to use a virtual network service endpoint.	Audit, Disabled	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Storage accounts should use customer-managed key for encryption</a>	Secure your blob and file storage account with greater flexibility using customer-managed keys. When you specify a customer-managed key, that key is used to protect and control access to the key that encrypts your data. Using customer-managed keys provides additional capabilities to control rotation of the key encryption key or cryptographically erase data.	Audit, Disabled	1.0.3
<a href="#">Storage accounts should use private link</a>	Azure Private Link lets you connect your virtual network to Azure services without a public IP address at the source or destination. The Private Link platform handles the connectivity between the consumer and services over the Azure backbone network. By mapping private endpoints to your storage account, data leakage risks are reduced. Learn more about private links at - <a href="https://aka.ms/azureprivatelinkoverview">https://aka.ms/azureprivatelinkoverview</a>	AuditIfExists, Disabled	2.0.0
<a href="#">Table Storage should use customer-managed key for encryption</a>	Secure your table storage with greater flexibility using customer-managed keys. When you specify a customer-managed key, that key is used to protect and control access to the key that encrypts your data. Using customer-managed keys provides additional capabilities to control rotation of the key encryption key or cryptographically erase data.	Audit, Deny, Disabled	1.0.0

## Microsoft.StorageCache

NAME (AZURE PORTAL)	DESCRIPTION	EFFECT(S)	VERSION (GITHUB)
<a href="#">HPC Cache accounts should use customer-managed key for encryption</a>	Manage encryption at rest of Azure HPC Cache with customer-managed keys. By default, customer data is encrypted with service-managed keys, but customer-managed keys are commonly required to meet regulatory compliance standards. Customer-managed keys enable the data to be encrypted with an Azure Key Vault key created and owned by you. You have full control and responsibility for the key lifecycle, including rotation and management.	Audit, Disabled, Deny	<a href="#">2.0.0</a>

## Microsoft.StorageSync

NAME (AZURE PORTAL)	DESCRIPTION	EFFECT(S)	VERSION (GITHUB)
<a href="#">Azure File Sync should use private link</a>	Creating a private endpoint for the indicated Storage Sync Service resource allows you to address your Storage Sync Service resource from within the private IP address space of your organization's network, rather than through the internet-accessible public endpoint. Creating a private endpoint by itself does not disable the public endpoint.	AuditIfNotExists, Disabled	<a href="#">1.0.0</a>
<a href="#">Configure Azure File Sync with private endpoints</a>	A private endpoint is deployed for the indicated Storage Sync Service resource. This enables you to address your Storage Sync Service resource from within the private IP address space of your organization's network, rather than through the internet-accessible public endpoint. The existence of one or more private endpoints by themselves does not disable the public endpoint.	DeployIfNotExists, Disabled	<a href="#">1.0.0</a>

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Modify - Configure Azure File Sync to disable public network access</a>	The Azure File Sync's internet-accessible public endpoint are disabled by your organizational policy. You may still access the Storage Sync Service via its private endpoint(s).	Modify, Disabled	1.0.0
<a href="#">Public network access should be disabled for Azure File Sync</a>	Disabling the public endpoint allows you to restrict access to your Storage Sync Service resource to requests destined to approved private endpoints on your organization's network. There is nothing inherently insecure about allowing requests to the public endpoint, however, you may wish to disable it to meet regulatory, legal, or organizational policy requirements. You can disable the public endpoint for a Storage Sync Service by setting the incomingTrafficPolicy of the resource to AllowVirtualNetworksOnly.	Audit, Deny, Disabled	1.0.0

## Microsoft.ClassicStorage

NAME (AZURE PORTAL)	DESCRIPTION	EFFECT(S)	VERSION (GITHUB)
<a href="#">Storage accounts should be migrated to new Azure Resource Manager resources</a>	Use new Azure Resource Manager for your storage accounts to provide security enhancements such as: stronger access control (RBAC), better auditing, Azure Resource Manager based deployment and governance, access to managed identities, access to key vault for secrets, Azure AD-based authentication and support for tags and resource groups for easier security management	Audit, Deny, Disabled	1.0.0

## Next steps

- See the built-ins on the [Azure Policy GitHub repo](#).
- Review the [Azure Policy definition structure](#).

- Review [Understanding policy effects](#).

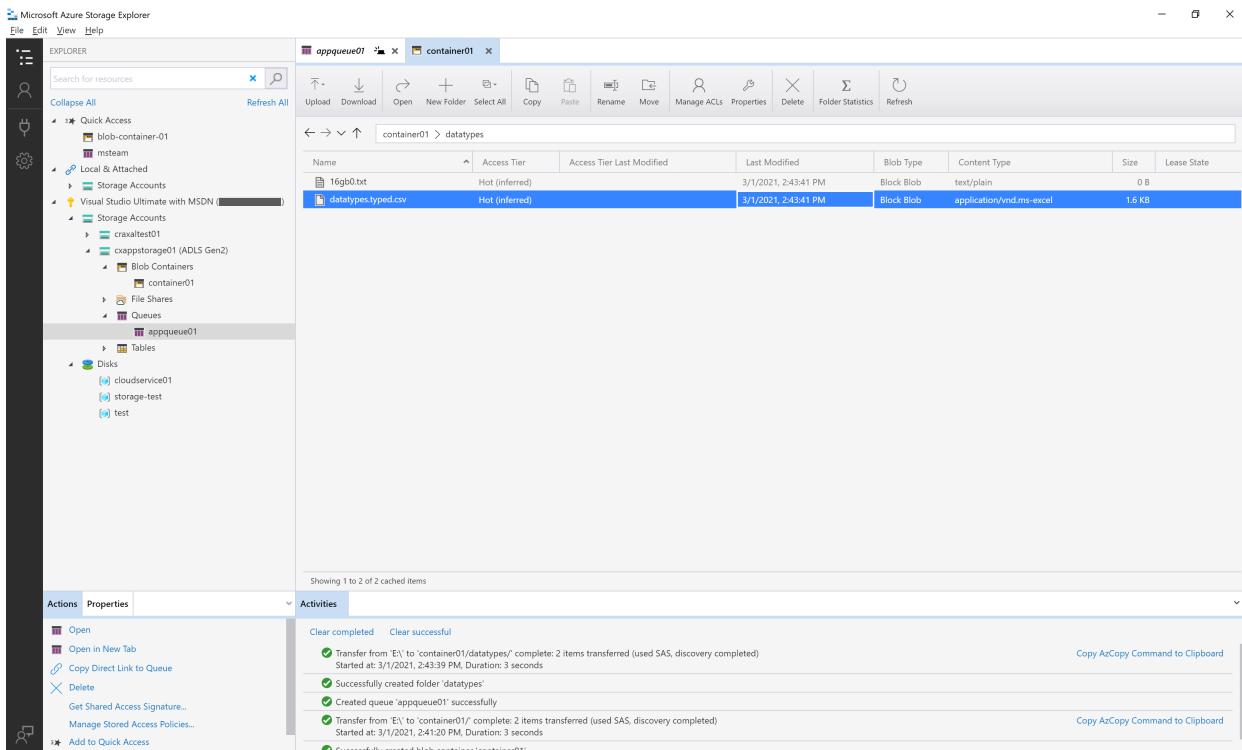
# Get started with Storage Explorer

8/22/2022 • 8 minutes to read • [Edit Online](#)

## Overview

Microsoft Azure Storage Explorer is a standalone app that makes it easy to work with Azure Storage data on Windows, macOS, and Linux.

In this article, you'll learn several ways of connecting to and managing your Azure storage accounts.



## Prerequisites

- [Windows](#)
- [macOS](#)
- [Linux](#)

The following versions of Windows support Storage Explorer:

- Windows 11
- Windows 10
- Windows 8
- Windows 7

For all versions of Windows, Storage Explorer requires .NET Framework 4.7.2 at a minimum.

## Download and install

To download and install Storage Explorer, see [Azure Storage Explorer](#).

## Connect to a storage account or service

Storage Explorer provides several ways to connect to Azure resources:

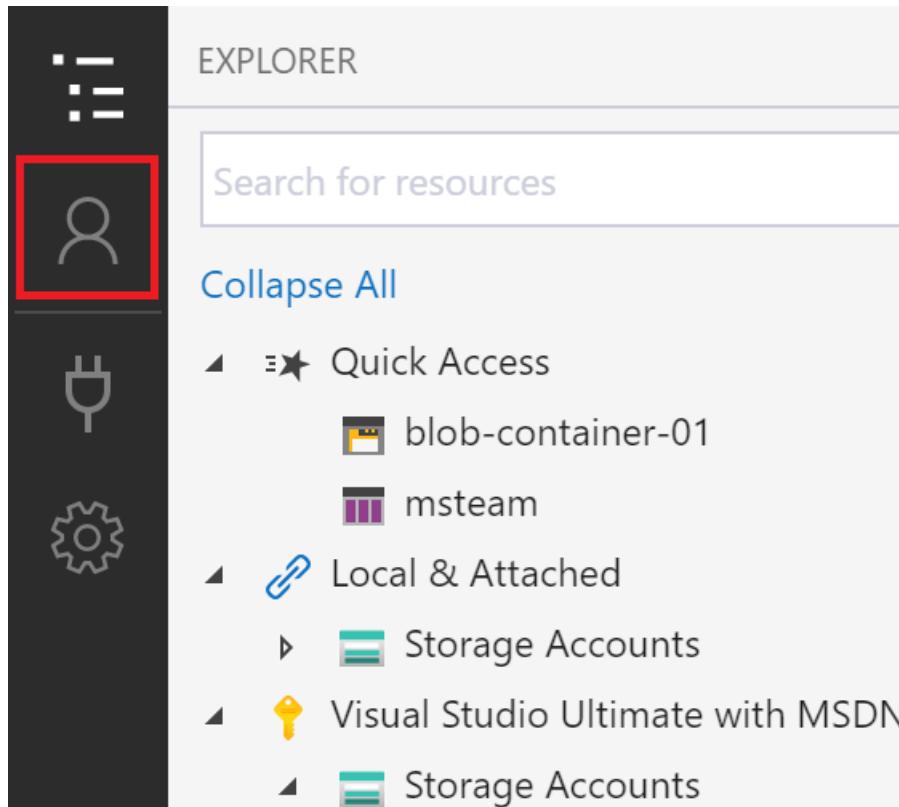
- [Sign in to Azure to access your subscriptions and their resources](#)
- [Attach to an individual Azure Storage resource](#)

## Sign in to Azure

### NOTE

To fully access resources after you sign in, Storage Explorer requires both management (Azure Resource Manager) and data layer permissions. This means that you need Azure Active Directory (Azure AD) permissions to access your storage account, the containers in the account, and the data in the containers. If you have permissions only at the data layer, consider choosing the **Sign in using Azure Active Directory (Azure AD)** option when attaching to a resource. For more information about the specific permissions Storage Explorer requires, see the [Azure Storage Explorer troubleshooting guide](#).

1. In Storage Explorer, select **View > Account Management** or select the **Manage Accounts** button.



2. **ACCOUNT MANAGEMENT** now displays all the Azure accounts you're signed in to. To connect to another account, select **Add an account....**
3. The **Connect to Azure Storage** dialog opens. In the **Select Resource** panel, select **Subscription**.

## Select Resource

Select Resource > Authenticate > Connect

What kind of Azure resource do you want to connect to?

-  **Subscription**  
Sign in to Azure to access storage resources such as blobs, files, queues, and tables under subscriptions you have access to.
-  **Storage account**  
Attach to a Storage account.
-  **Blob container**  
Attach to a Blob container.
-  **ADLS Gen2 container or directory**  
Attach to an ADLS Gen2 container or directory.
-  **File share**  
Attach to a File share.
-  **Queue**  
Attach to a queue.
-  **Table**  
Attach to a table.
-  **Local storage emulator**  
Attach to resources managed by a storage emulator running on your local machine.

[Cancel](#)

4. In the **Select Azure Environment** panel, select an Azure environment to sign in to. You can sign in to global Azure, a national cloud or an Azure Stack instance. Then select **Next**.

## Select Azure Environment

Select Resource > Select Azure Environment > Sign In

Which Azure environment will you use to sign in?

- Azure
- Azure China
- Azure Germany
- Azure US Government
- Custom Environment:

test

[Manage custom environments...](#)

[Back](#)

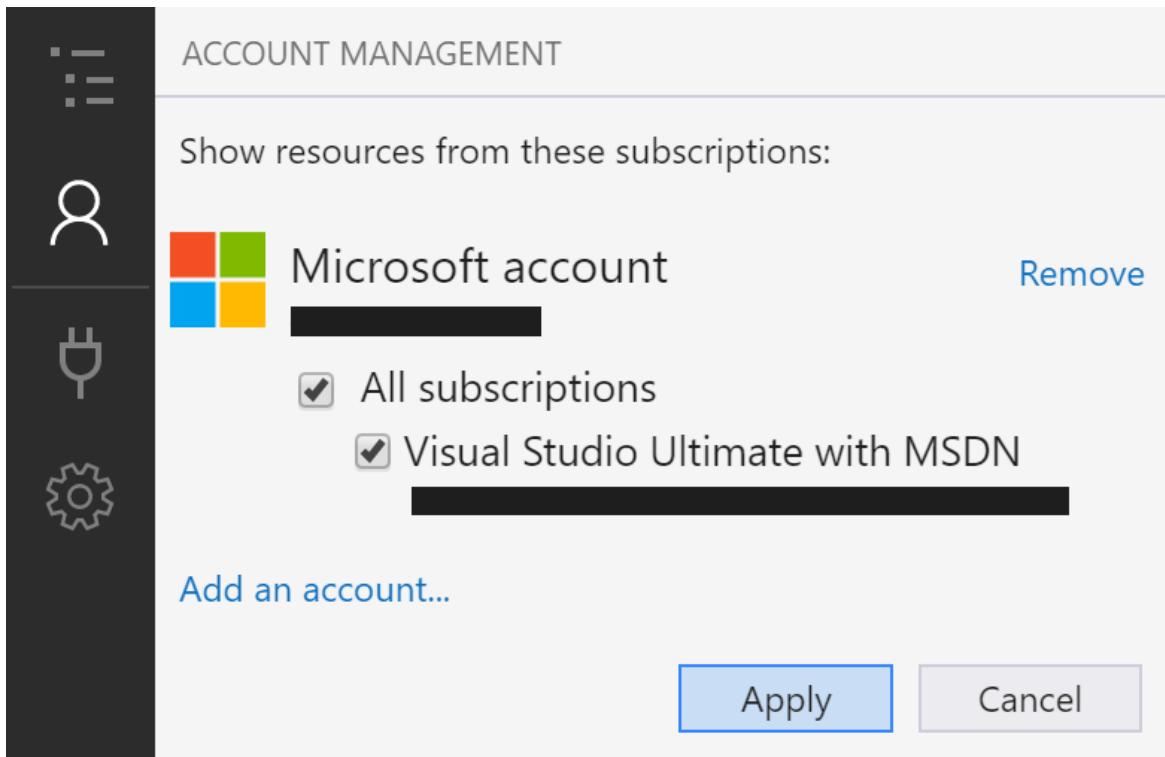
[Next](#)

[Cancel](#)

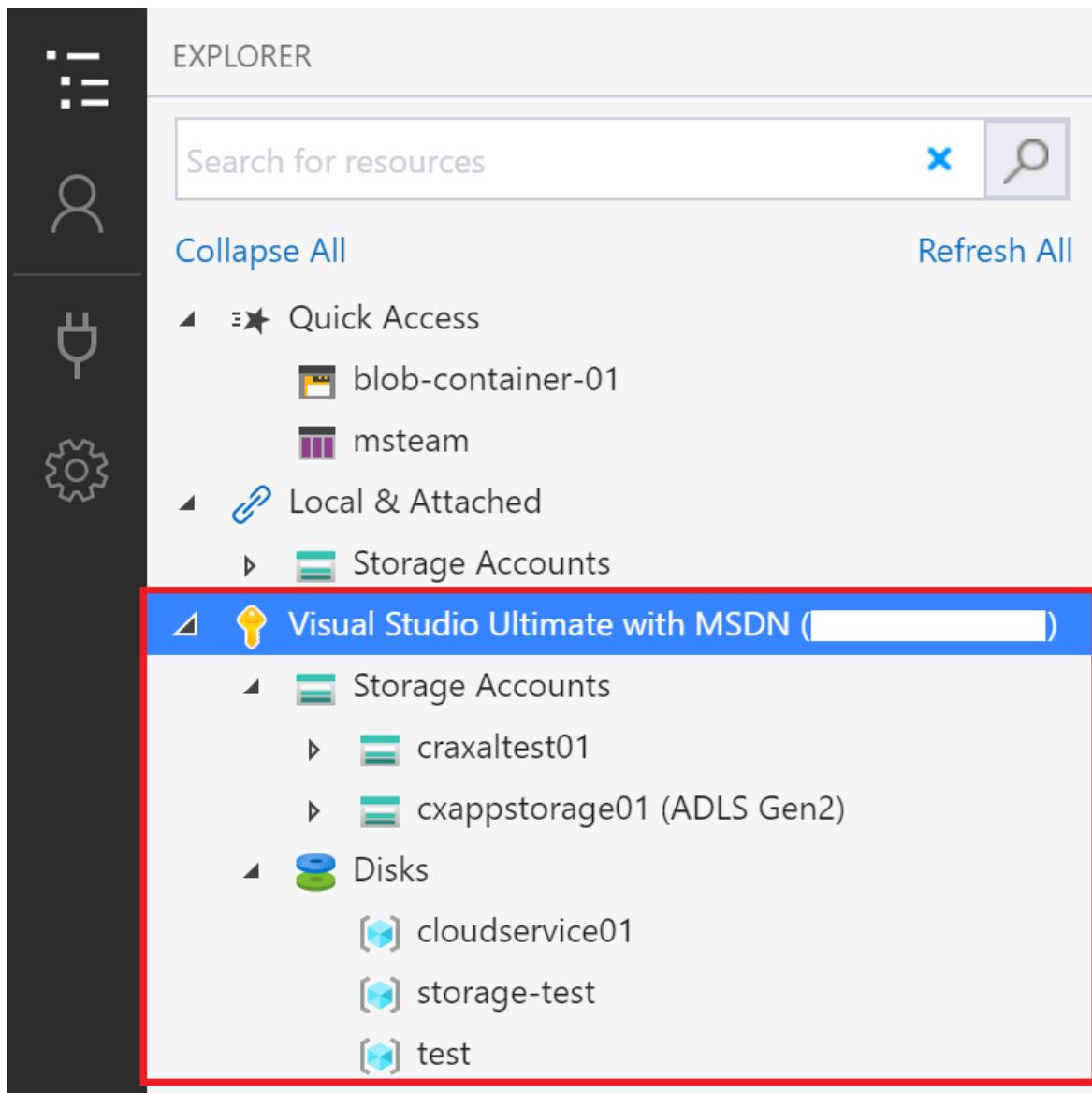
**TIP**

For more information about Azure Stack, see [Connect Storage Explorer to an Azure Stack subscription or storage account](#).

5. Storage Explorer will open a webpage for you to sign in.
6. After you successfully sign in with an Azure account, the account and the Azure subscriptions associated with that account appear under **ACCOUNT MANAGEMENT**. Select the Azure subscriptions that you want to work with, and then select **Apply**.



7. EXPLORER displays the storage accounts associated with the selected Azure subscriptions.



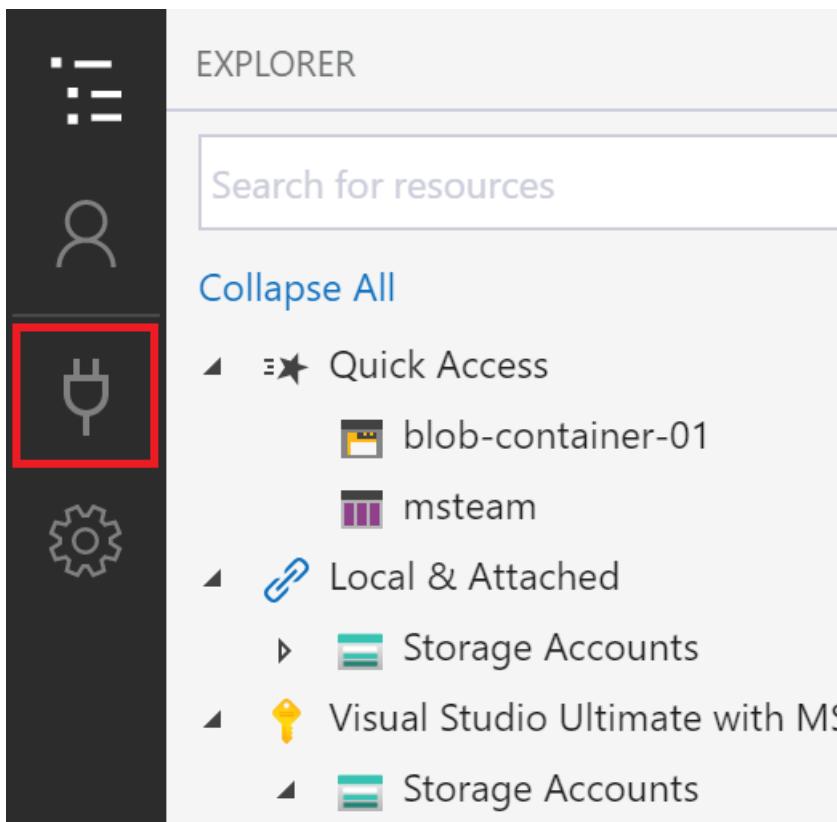
#### Attach to an individual resource

Storage Explorer lets you connect to individual resources, such as an Azure Data Lake Storage Gen2 container, using various authentication methods. Some authentication methods are only supported for certain resource types.

RESOURCE TYPE	AZURE AD	ACCOUNT NAME AND KEY	SHARED ACCESS SIGNATURE (SAS)	PUBLIC (ANONYMOUS)
Storage accounts	Yes	Yes	Yes (connection string or URL)	No
Blob containers	Yes	No	Yes (URL)	Yes
Gen2 containers	Yes	No	Yes (URL)	Yes
Gen2 directories	Yes	No	Yes (URL)	Yes
File shares	No	No	Yes (URL)	No
Queues	Yes	No	Yes (URL)	No
Tables	Yes	No	Yes (URL)	No

Storage Explorer can also connect to a [local storage emulator](#) using the emulator's configured ports.

To connect to an individual resource, select the **Connect** button in the left-hand toolbar. Then follow the instructions for the resource type you want to connect to.



When a connection to a storage account is successfully added, a new tree node will appear under **Local & Attached > Storage Accounts**.

For other resource types, a new node is added under **Local & Attached > Storage Accounts > (Attached Containers)**. The node will appear under a group node matching its type. For example, a new connection to an Azure Data Lake Storage Gen2 container will appear under **Blob Containers**.

If Storage Explorer couldn't add your connection, or if you can't access your data after successfully adding the connection, see the [Azure Storage Explorer troubleshooting guide](#).

The following sections describe the different authentication methods you can use to connect to individual resources.

#### Azure AD

Storage Explorer can use your Azure account to connect to the following resource types:

- Blob containers
- Azure Data Lake Storage Gen2 containers
- Azure Data Lake Storage Gen2 directories
- Queues

Azure AD is the preferred option if you have data layer access to your resource but no management layer access.

1. Sign in to at least one Azure account using the [steps described above](#).
2. In the **Select Resource** panel of the **Connect to Azure Storage** dialog, select **Blob container**, **ADLS Gen2 container**, or **Queue**.
3. Select **Sign in using Azure Active Directory (Azure AD)** and select **Next**.
4. Select an Azure account and tenant. The account and tenant must have access to the Storage resource you

want to attach to. Select **Next**.

5. Enter a display name for your connection and the URL of the resource. Select **Next**.
6. Review your connection information in the **Summary** panel. If the connection information is correct, select **Connect**.

#### Account name and key

Storage Explorer can connect to a storage account using the storage account's name and key.

You can find your account keys in the [Azure portal](#). Open your storage account page and select **Settings > Access keys**.

1. In the **Select Resource** panel of the **Connect to Azure Storage** dialog, select **Storage account**.
2. Select **Account name and key** and select **Next**.
3. Enter a display name for your connection, the name of the account, and one of the account keys. Select the appropriate Azure environment. Select **Next**.
4. Review your connection information in the **Summary** panel. If the connection information is correct, select **Connect**.

#### Shared access signature (SAS) connection string

Storage Explorer can connect to a storage account using a connection string with a Shared Access Signature (SAS). A SAS connection string looks like this:

```
SharedAccessSignature=sv=2020-04-08&ss=btqf&srt=sco&st=2021-03-02T00%3A22%3A19Z&se=2020-03-03T00%3A22%3A19Z&sp=r&sig=fFFpX%2F5tzqmmFfaL0wRffH1hfFFLn6zJuy1T6yhOo%2FY%3F;
BlobEndpoint=https://contoso.blob.core.windows.net/;
FileEndpoint=https://contoso.file.core.windows.net/;
QueueEndpoint=https://contoso.queue.core.windows.net/;
TableEndpoint=https://contoso.table.core.windows.net/;
```

1. In the **Select Resource** panel of the **Connect to Azure Storage** dialog, select **Storage account**.
2. Select **Shared access signature (SAS)** and select **Next**.
3. Enter a display name for your connection and the SAS connection string for the storage account. Select **Next**.
4. Review your connection information in the **Summary** panel. If the connection information is correct, select **Connect**.

#### Shared access signature (SAS) URL

Storage Explorer can connect to the following resource types using a SAS URI:

- Blob container
- Azure Data Lake Storage Gen2 container or directory
- File share
- Queue
- Table

A SAS URI looks like this:

```
https://contoso.blob.core.windows.net/container01?sv=2020-04-08&st=2021-03-02T00%3A30%3A33Z&se=2020-03-03T00%3A30%3A33Z&sr=c&sp=r&sig=z9VFDWffrV6FXU51T8b8HVfipZP0pYOFLXuQw6wfkFY%3F
```

1. In the **Select Resource** panel of the **Connect to Azure Storage** dialog, select the resource you want to connect to.
2. Select **Shared access signature (SAS)** and select **Next**.
3. Enter a display name for your connection and the SAS URI for the resource. Select **Next**.
4. Review your connection information in the **Summary** panel. If the connection information is correct, select

## Connect.

### Local storage emulator

Storage Explorer can connect to an Azure Storage emulator. Currently, there are two supported emulators:

- [Azure Storage Emulator](#) (Windows only)
- [Azurite](#) (Windows, macOS, or Linux)

If your emulator is listening on the default ports, you can use the **Local & Attached > Storage Accounts > Emulator - Default Ports** node to access your emulator.

If you want to use a different name for your connection, or if your emulator isn't running on the default ports:

1. Start your emulator.

#### IMPORTANT

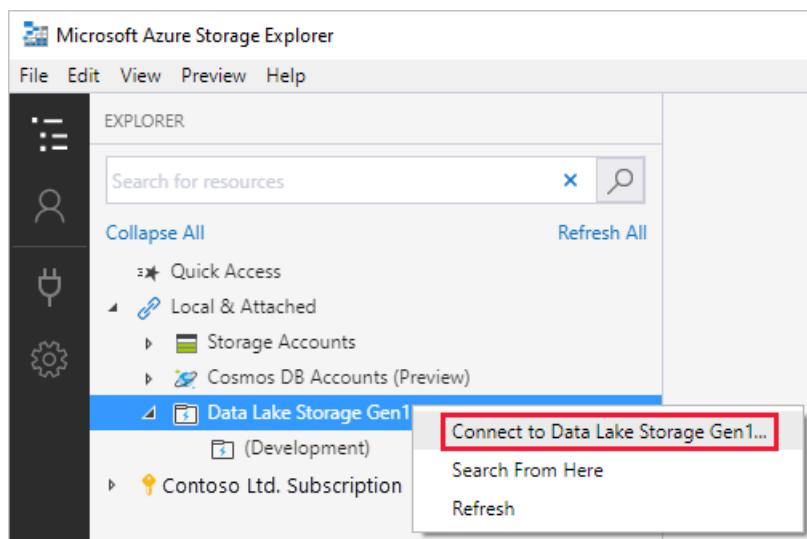
Storage Explorer doesn't automatically start your emulator. You must start it manually.

2. In the **Select Resource** panel of the **Connect to Azure Storage** dialog, select **Local storage emulator**.
3. Enter a display name for your connection and the port number for each emulated service you want to use. If you don't want to use to a service, leave the corresponding port blank. Select **Next**.
4. Review your connection information in the **Summary** panel. If the connection information is correct, select **Connect**.

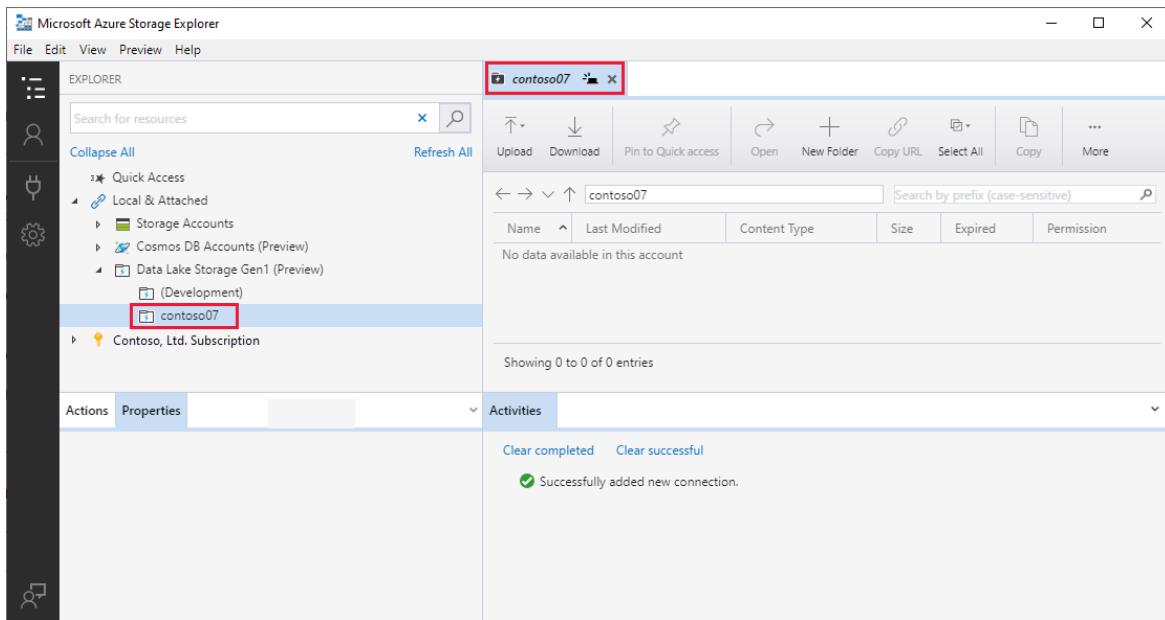
### Connect to Azure Data Lake Store by URI

You can access a resource that's not in your subscription. You need someone who has access to that resource to give you the resource URI. After you sign in, connect to Data Lake Store by using the URI. To connect, follow these steps:

1. Under **EXPLORER**, expand **Local & Attached**.
2. Right-click **Data Lake Storage Gen1**, and select **Connect to Data Lake Storage Gen1**.



3. Enter the URI, and then select **OK**. Your Data Lake Store appears under **Data Lake Storage**.

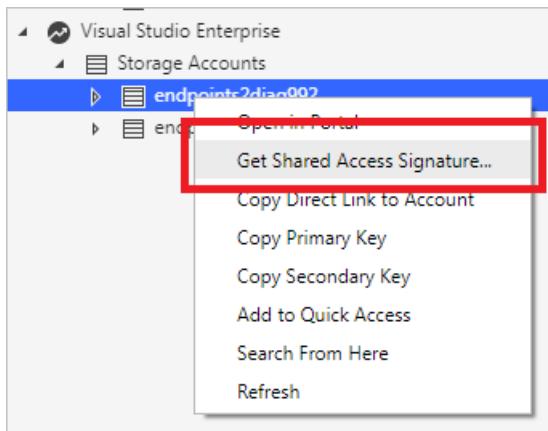


This example uses Data Lake Storage Gen1. Azure Data Lake Storage Gen2 is now available. For more information, see [What is Azure Data Lake Storage Gen1](#).

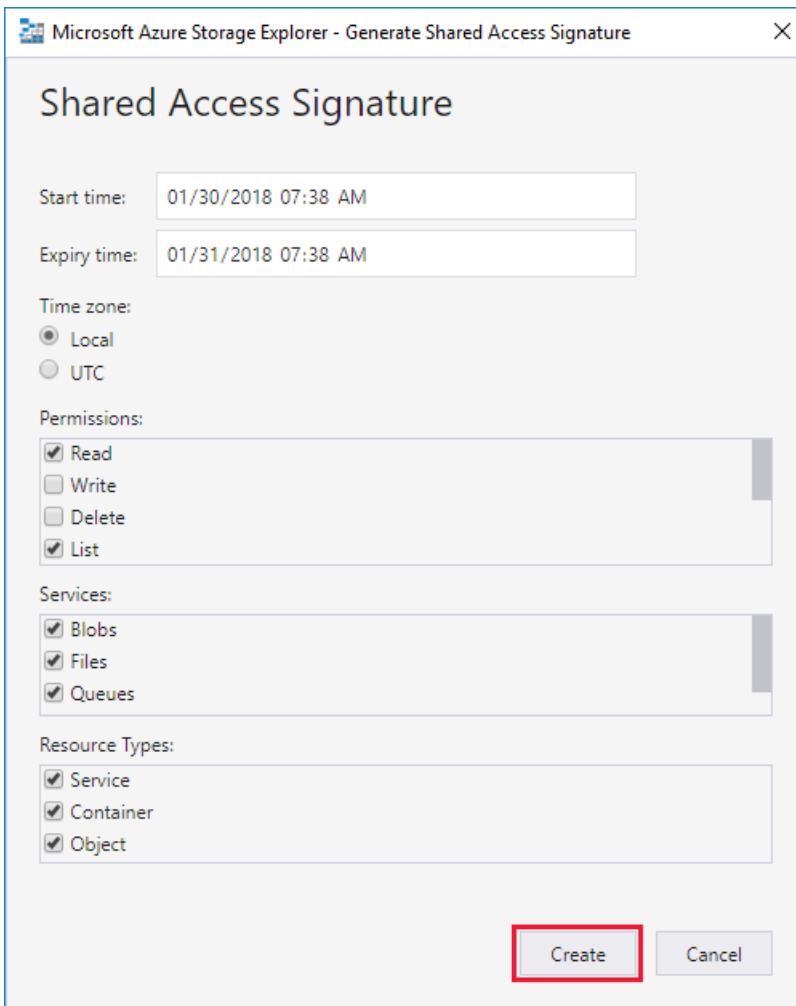
## Generate a shared access signature in Storage Explorer

### Account level shared access signature

1. Right-click the storage account you want share, and then select Get Shared Access Signature.



2. In Shared Access Signature, specify the time frame and permissions you want for the account, and then select Create.



3. Copy either the **Connection string** or the raw **Query string** to your clipboard.

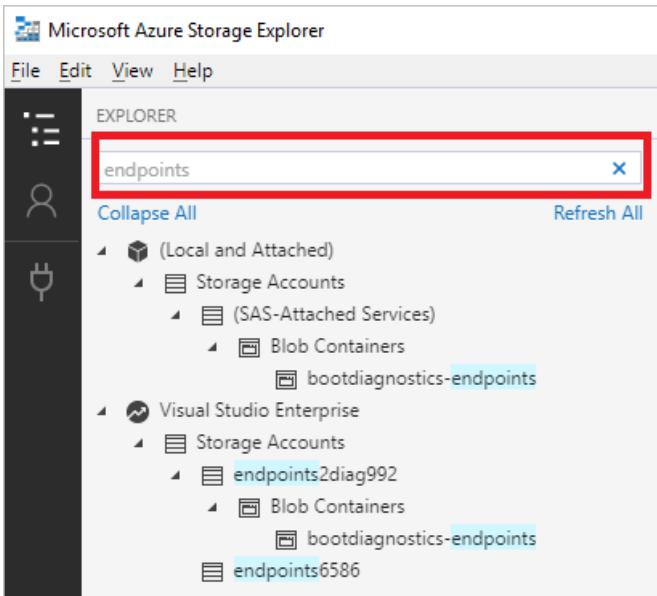
#### Service level shared access signature

You can get a shared access signature at the service level. For more information, see [Get the SAS for a blob container](#).

## Search for storage accounts

To find a storage resource, you can search in the **EXPLORER** pane.

As you enter text in the search box, Storage Explorer displays all resources that match the search value you've entered up to that point. This example shows a search for **endpoints**:



#### NOTE

To speed up your search, use **Account Management** to deselect any subscriptions that don't contain the item you're searching for. You can also right-click a node and select **Search From Here** to start searching from a specific node.

## Next steps

- [Manage Azure Blob storage resources with Storage Explorer](#)
- [Manage Azure Data Lake Store resources with Storage Explorer](#)

# Sign in to Storage Explorer

8/22/2022 • 3 minutes to read • [Edit Online](#)

Sign-in is the recommended way to access your Azure storage resources with Storage Explorer. By signing in you take advantage of Azure AD backed permissions, such as RBAC and Gen2 POSIX ACLs.

## How to sign in

To sign in to Storage Explorer, open the **Connect dialog**. You can open the **Connect dialog** either from the left-hand vertical toolbar, or by clicking on **Add account...** on the **Account Panel**.

Once you have the dialog open, choose **Subscription** as the type of resource you want to connect to and click **Next**.

You now need to choose what Azure environment you want to sign into. You can pick from any of the known environments, such as Azure or Azure China, or you can add your own environment. Once you have your environment selected, click **Next**.

At this point, your OS' **default web browser** will launch and a sign-in page will be opened. For best results, leave this browser window open as long as you're using Storage Explorer or at least until you've performed all expected MFA. When you have finished signing in, you can return to Storage Explorer.

## Managing accounts

You can manage and remove Azure accounts that you've signed into from the **Account Panel**. You can open the **Account Panel** by clicking on the **Manage Accounts** button on the left-hand vertical toolbar.

In the **Account Panel** you'll see any accounts that you have signed into. Under each account will be:

- The tenants the account belongs to
- For each tenant, the subscriptions you have access to

By default, Storage Explorer only signs you into your home tenant. If you want to view subscriptions and resources from another tenant, you'll need to activate that tenant. To activate a tenant, check the checkbox next to it. Once you're done working with a tenant, you can uncheck its checkbox to deactivate it. You cannot deactivate your home tenant.

After activating a tenant, you may need to reenter your credentials before Storage Explorer can load subscriptions or access resources from the tenant. Having to reenter your credentials usually happens because of a conditional access (CA) policy such as multi-factor authentication (MFA). And even though you may have already performed MFA for another tenant, you might still have to do it again. To reenter your credentials, simply click on **Reenter credentials....** You can also click on **Error details...** to see exactly why subscriptions failed to load.

Once your subscriptions have loaded, you can choose which ones you want to filter in/out by checking or unchecking their checkboxes.

If you want to remove your entire Azure account, then click on the **Remove** next to the account.

## Changing where sign-in happens

By default sign-in will happen in your OS' **default web browser**. Signing-in with your default web browser streamlines how you access resources secured via CA policies, such as MFA. If for some reason signing in with

your OS' **default web browser** isn't working, you can change where or how Storage Explorer performs sign-in.

Under **Settings** (gear icon on the left) > **Application** > **Sign-in**, look for the **Sign in with** setting. There are three options:

- **Default Web Browser**: sign-in will happen in your OS' **default web browser**. This option is recommended.
- **Integrated Sign-In**: sign-in will happen in a Storage Explorer window. This option may be useful if you're having issues using your **default web browser** to sign in.
- **Device Code Flow**: Storage Explorer will give you a code to enter into a browser window. This option isn't recommended. Device code flow isn't compatible with many CA policies.

## Troubleshooting sign-in issues

If you're having trouble signing in, or are having issues with an Azure account after signing in, refer to the [sign in section of the Storage Explorer troubleshooting guide](#).

## Next steps

- [Manage Azure Blob storage resources with Storage Explorer](#)
- [Troubleshoot sign in issues](#)

# Network connections in Storage Explorer

8/22/2022 • 5 minutes to read • [Edit Online](#)

When not connecting to a local emulator, Storage Explorer uses your network to make requests to your storage resources and other Azure and Microsoft services.

## Hostnames accessed by Storage Explorer

Storage Explorer makes requests to various endpoints while in use. The following list details common hostnames that Storage Explorer makes requests to:

- ARM endpoints:
  - `management.azure.com` (global Azure)
  - `management.chinacloudapi.cn` (Azure China)
  - `management.microsoftazure.de` (Azure Germany)
  - `management.usgovcloudapi.net` (Azure US Government)
- Login endpoints:
  - `login.microsoftonline.com` (global Azure)
  - `login.chinacloudapi.cn` (Azure China)
  - `login.microsoftonline.de` (Azure Germany)
  - `login.microsoftonline.us` (Azure US Government)
- Graph endpoints:
  - `graph.windows.net` (global Azure)
  - `graph.chinacloudapi.cn` (Azure China)
  - `graph.cloudapi.de` (Azure Germany)
  - `graph.windows.net` (Azure US Government)
- Azure Storage endpoints:
  - `(blob|file|queue|table|dfs).core.windows.net` (global Azure)
  - `(blob|file|queue|table|dfs).core.chinacloudapi.cn` (Azure China)
  - `(blob|file|queue|table|dfs).core.cloudapi.de` (Azure Germany)
  - `(blob|file|queue|table|dfs).core.usgovcloudapi.net` (Azure US Government)
- Storage Explorer updating: `storageexplorerpublish.blob.core.windows.net`
- Microsoft link forwarding:
  - `aka.ms`
  - `go.microsoft.com`
- Any custom domains, private links, or Azure Stack instance-specific endpoints, that your resources are behind
- Remote emulator hostnames

## Proxy sources

Storage Explorer has several options for how/where it can source the information needed to connect to your proxy. To change which option is being used, go to **Settings** (gear icon on the left vertical toolbar) > **Application** > **Proxy**. Once you are at the proxy section of settings, you can select how/where you want Storage Explorer to source your proxy settings:

- Do not use proxy

- Use environment variables
- Use app proxy settings
- Use system proxy (preview)

### Do not use proxy

When this option is selected, Storage Explorer won't make any attempt to connect to a proxy. Do not use proxy is the default option.

### Use environment variables

When this option is selected, Storage Explorer will look for proxy information from specific environment variables. These variables are:

- `HTTP_PROXY`
- `HTTPS_PROXY`

If both variables are defined, then Storage Explorer will source proxy information from `HTTPS_PROXY`.

The value of these environment variables must be a url of the format:

```
(http|https)://(username:password@)<hostname>:<port>
```

Only the protocol (`http|https`), and hostname are required. If you have a username, you don't have to supply a password.

### Use app proxy settings

When this option is selected, Storage Explorer will use the in app proxy settings. These settings include:

- Protocol
- Hostname
- Port
- Do/do not use credentials
- Credentials

All settings other than credentials can be managed from either:

- **Settings** (gear icon on the left vertical toolbar) > **Application** > **Proxy** > **Use credentials**.
- The Proxy Settings dialog (**Edit** > **Configure Proxy**).

To set credentials, you must go to the Proxy Settings dialog (**Edit** > **Configure Proxy**).

### Use system proxy (preview)

When this option is selected, Storage Explorer will use your OS proxy settings. More specifically, it will result in network calls being made using the Chromium networking stack. The Chromium networking stack is much more robust than the NodeJS networking stack normally used by Storage Explorer. Here's a snippet from [Chromium's documentation](#) on what all it can do:

The Chromium network stack uses the system network settings so that users and administrators can control the network settings of all applications easily. The network settings include:

- proxy settings
- SSL/TLS settings
- certificate revocation check settings
- certificate and private key stores

If your proxy server requires credentials, and those credentials aren't configured at your OS settings, you'll need

to enable usage of and set your credentials within in Storage Explorer. You can toggle use of credentials from either:

- **Settings** (gear icon on the left vertical toolbar) > **Application** > **Proxy** > **Use credentials**.
- The Proxy Settings dialog (**Edit** > **Configure Proxy**).

To set credentials, you must go to the Proxy Settings dialog (**Edit** > **Configure Proxy**).

This option is in preview because not all features currently support system proxy. See [features that do not support system proxy](#) for a complete list of features which do not support it. When system proxy is enabled, features that don't support system proxy won't make any attempt to connect to a proxy.

If you come across an issue while using system proxy with a supported feature, [open an issue on GitHub](#).

## Proxy server authentication

If you have configured Storage Explorer to source proxy settings from **environment variables** or **app proxy settings**, then only proxy servers that use basic authentication are supported.

If you have configured Storage Explorer to use **system proxy**, then proxy servers that use any of the following authentication methods are supported:

- Basic
- Digest
- NTLM
- Negotiate

## Which proxy source should I choose?

If you're using features not listed [here](#), then you should first try using **system proxy**. If you come across an issue while using system proxy with a supported feature, [open an issue on GitHub](#).

If you're using features that don't support system proxy, then **app settings** is probably the next best option. The GUI-based experience for configuring the proxy configuration helps reduce the chance of entering your proxy information correctly. However, if you already have proxy environment variables configured, then it might be better to use **environment variables**.

## AzCopy proxy usage

Storage Explorer uses AzCopy for most data transfers operations. AzCopy is written using a different set of technologies than Storage Explorer and therefore has a slightly different set of proxy capabilities.

If Storage Explorer is configured to **do not use proxy** or to use **system proxy**, then AzCopy will be told to use its own autodetect proxy features to determine if and how it should make requests to a proxy. If you have configured Storage Explorer to source proxy settings from **environment variables** or **app proxy settings** though, then Storage Explorer will tell AzCopy to use the same proxy settings.

If you don't want AzCopy to use proxy no matter what, then you can disable proxy usage by toggling **Settings** (gear icon on the left vertical toolbar) > **Transfers** > **AzCopy** > **Disable AzCopy Proxy Usage**.

Currently, AzCopy only supports proxy servers that use **basic authentication**.

## SSL certificates

By default, Storage Explorer uses the NodeJS networking stack. NodeJS ships with a predefined list of trusted SSL certificates. Some networking technologies, such as proxy servers or anti-virus software, inject their own SSL certificates into network traffic. These certificates are often not present in NodeJS' certificate list. NodeJS

won't trust responses that contain such a certificate. When NodeJS doesn't trust a response, then Storage Explorer will receive an error.

You have multiple options for resolving such errors:

- Use [system proxy](#) as your proxy source.
- Import a copy of the SSL certificate/s causing the error/s.
- Disable SSL certificate. (**not recommended**)

## Features that do not support system proxy

The following is a list of features that do not support [system proxy](#):

- Storage Account Features
  - Setting default access tier
- Table Features
  - Manage access policies
  - Configure CORS
  - Generate SAS
  - Copy & Paste Table
  - Clone Table
- All ADLS Gen1 features

## Next steps

- [Troubleshoot proxy issues](#)
- [Troubleshoot certificate issues](#)

# Azure Storage Explorer troubleshooting guide

8/22/2022 • 24 minutes to read • [Edit Online](#)

Microsoft Azure Storage Explorer is a standalone app that makes it easy to work with Azure Storage data on Windows, macOS, and Linux. The app can connect to storage accounts hosted on Azure, national clouds, and Azure Stack.

This guide summarizes solutions for issues that are commonly seen in Storage Explorer.

## Azure RBAC permissions issues

Azure role-based access control ([Azure RBAC](#)) enables highly granular access management of Azure resources by combining sets of permissions into *roles*. Here are some strategies to get Azure RBAC working optimally in Storage Explorer.

### How do I access my resources in Storage Explorer?

If you're having problems accessing storage resources through Azure RBAC, you might not have been assigned the appropriate roles. The following sections describe the permissions Storage Explorer currently requires for access to your storage resources. Contact your Azure account admin if you're not sure you have the appropriate roles or permissions.

#### "Read: List/Get Storage Account(s)" permissions issue

You must have permission to list storage accounts. To get this permission, you must be assigned the *Reader* role.

#### List storage account keys

Storage Explorer can also use account keys to authenticate requests. You can get access to account keys through more powerful roles, such as the *Contributor* role.

#### NOTE

Access keys grant unrestricted permissions to anyone who holds them. As a result, we don't recommend that you hand out these keys to account users. If you need to revoke access keys, you can regenerate them from the [Azure portal](#).

#### Data roles

You must be assigned at least one role that grants access to read data from resources. For example, if you want to list or download blobs, you'll need at least the *Storage Blob Data Reader* role.

### Why do I need a management layer role to see my resources in Storage Explorer?

Azure Storage has two layers of access: *management* and *data*. Subscriptions and storage accounts are accessed through the management layer. Containers, blobs, and other data resources are accessed through the data layer. For example, if you want to get a list of your storage accounts from Azure, you send a request to the management endpoint. If you want a list of blob containers in an account, you send a request to the appropriate service endpoint.

Azure roles can grant you permissions for management or data layer access. The Reader role, for example, grants read-only access to management layer resources.

Strictly speaking, the Reader role provides no data layer permissions and isn't necessary for accessing the data layer.

Storage Explorer makes it easy to access your resources by gathering the necessary information to connect to your Azure resources. For example, to display your blob containers, Storage Explorer sends a "list containers"

request to the blob service endpoint. To get that endpoint, Storage Explorer searches the list of subscriptions and storage accounts you have access to. To find your subscriptions and storage accounts, Storage Explorer also needs access to the management layer.

If you don't have a role that grants any management layer permissions, Storage Explorer can't get the information it needs to connect to the data layer.

### What if I can't get the management layer permissions I need from my admin?

If you want to access blob containers, Azure Data Lake Storage Gen2 containers or directories, or queues, you can attach to those resources by using your Azure credentials.

1. Open the **Connect** dialog.
2. Select the resource type you want to connect to.
3. Select **Sign in using Azure Active Directory (Azure AD)** and select **Next**.
4. Select the user account and tenant associated with the resource you're attaching to. Select **Next**.
5. Enter the URL to the resource, and enter a unique display name for the connection. Select **Next** and then select **Connect**.

For other resource types, we don't currently have an Azure RBAC-related solution. As a workaround, you can request a shared access signature URL and then attach to your resource:

1. Open the **Connect** dialog.
2. Select the resource type you want to connect to.
3. Select **Shared access signature (SAS)** and select **Next**.
4. Enter the shared access signature URL you received and enter a unique display name for the connection. Select **Next** and then select **Connect**.

For more information on how to attach to resources, see [Attach to an individual resource](#).

### Recommended Azure built-in roles

There are several Azure built-in roles that can provide the permissions needed to use Storage Explorer. Some of those roles are:

- **Owner**: Manage everything, including access to resources.
- **Contributor**: Manage everything, excluding access to resources.
- **Reader**: Read and list resources.
- **Storage Account Contributor**: Full management of storage accounts.
- **Storage Blob Data Owner**: Full access to Azure Storage blob containers and data.
- **Storage Blob Data Contributor**: Read, write, and delete Azure Storage containers and blobs.
- **Storage Blob Data Reader**: Read and list Azure Storage containers and blobs.

#### NOTE

The Owner, Contributor, and Storage Account Contributor roles grant account key access.

## SSL certificate issues

This section discusses SSL certificate issues.

### Understand SSL certificate issues

Make sure you've read the [SSL certificates section](#) in the Storage Explorer networking documentation before you continue.

### Use system proxy

If you're only using features that support the **use system proxy** setting, try using that setting. To read more about the **system proxy** setting, see [Network connections in Storage Explorer](#).

## Import SSL certificates

If you have a copy of the self-signed certificates, you can instruct Storage Explorer to trust them:

1. Obtain a Base-64 encoded X.509 (.cer) copy of the certificate.
2. Go to **Edit > SSL Certificates > Import Certificates**. Then use the file picker to find, select, and open the .cer file.

This issue might also occur if there are multiple certificates (root and intermediate). To fix this error, all certificates must be imported.

## Find SSL certificates

If you don't have a copy of the self-signed certificates, talk to your IT admin for help.

Follow these steps to find them:

1. Install OpenSSL:
  - **Windows**: Any of the light versions should be sufficient.
  - Mac and Linux: Should be included with your operating system.
2. Run OpenSSL:
  - Windows: Open the installation directory, select `/bin/`, and then double-click `openssl.exe`.
  - Mac and Linux: Run `openssl` from a terminal.
3. Run the command `openssl s_client -showcerts -connect <hostname>:443` for any of the Microsoft or Azure host names that your storage resources are behind. For more information, see this [list of host names that are frequently accessed by Storage Explorer](#).
4. Look for self-signed certificates. If the subject `("s:")` and issuer `("i:")` are the same, the certificate is most likely self-signed.
5. When you find the self-signed certificates, for each one, copy and paste everything from, and including, `-----BEGIN CERTIFICATE-----` to `-----END CERTIFICATE-----` into a new .cer file.
6. Open Storage Explorer and go to **Edit > SSL Certificates > Import Certificates**. Then use the file picker to find, select, and open the .cer files you created.

## Disable SSL certificate validation

If you can't find any self-signed certificates by following these steps, contact us through the feedback tool. You can also open Storage Explorer from the command line with the `--ignore-certificate-errors` flag. When opened with this flag, Storage Explorer ignores certificate errors. *This flag is not recommended.*

# Sign-in issues

This section discusses sign-in issues you might encounter.

## Understand sign-in

Make sure you've read the [Sign in to Storage Explorer](#) documentation before you continue.

## Frequently having to reenter credentials

Having to reenter credentials is most likely the result of Conditional Access policies set by your Azure Active Directory (Azure AD) admin. When Storage Explorer asks you to reenter credentials from the account panel, you should see an **Error details** link. Select it to see why Storage Explorer is asking you to reenter credentials. Conditional Access policy errors that require reentering of credentials might look something like these:

- The refresh token has expired.
- You must use multifactor authentication to access.
- Your admin made a configuration change.

To reduce the frequency of having to reenter credentials because of errors like the preceding ones, you'll need to talk to your Azure AD admin.

### Conditional access policies

If you have conditional access policies that need to be satisfied for your account, make sure you're using the **Default Web Browser** value for the **Sign in with** setting. For information on that setting, see [Changing where sign-in happens](#).

### Browser complains about HTTP redirect or insecure connection during sign-in

When Storage Explorer performs sign-in in your web browser, a redirect to `localhost` is done at the end of the sign-in process. Browsers sometimes raise a warning or error that the redirect is being performed with HTTP instead of HTTPS. Some browsers might also try to force the redirect to be performed with HTTPS. If either of these issues happen, depending on your browser, you have options:

- Ignore the warning.
- Add an exception for `localhost`.
- Disable force HTTPS, either globally or just for `localhost`.

If you can't do any of those options, you can also [change where sign-in happens](#) to integrated sign-in to avoid using your browser altogether.

### Unable to acquire token, tenant is filtered out

If you see an error message that says a token can't be acquired because a tenant is filtered out, you're trying to access a resource that's in a tenant you filtered out. To unfilter the tenant, go to the **Account Panel**. Make sure the checkbox for the tenant specified in the error is selected. For more information on filtering tenants in Storage Explorer, see [Managing accounts](#).

### Authentication library failed to start properly

If on startup you see an error message that says Storage Explorer's authentication library failed to start properly, make sure your installation environment meets all [prerequisites](#). Not meeting prerequisites is the most likely cause of this error message.

If you believe that your installation environment meets all prerequisites, [open an issue on GitHub](#). When you open your issue, make sure to include:

- Your OS.
- What version of Storage Explorer you're trying to use.
- If you checked the prerequisites.
- [Authentication logs](#) from an unsuccessful launch of Storage Explorer. Verbose authentication logging is automatically enabled after this type of error occurs.

### Blank window when you use integrated sign-in

If you chose to use **Integrated Sign-in** and you're seeing a blank sign-in window, you'll likely need to switch to a different sign-in method. Blank sign-in dialog boxes most often occur when an Active Directory Federation Services server prompts Storage Explorer to perform a redirect that's unsupported by Electron.

To change to a different sign-in method, change the **Sign in with** setting under **Settings > Application > Sign-in**. For information on the different types of sign-in methods, see [Changing where sign in happens](#).

### Reauthentication loop or UPN change

If you're in a reauthentication loop or have changed the UPN of one of your accounts, try these steps:

1. Open Storage Explorer.
2. Go to **Help > Reset**.
3. Make sure at least **Authentication** is selected. Clear other items you don't want to reset.
4. Select **Reset**.
5. Restart Storage Explorer and try to sign in again.

If you continue to have issues after you do a reset, try these steps:

1. Open Storage Explorer.
2. Remove all accounts and then close Storage Explorer.
3. Delete the `./identityService` folder from your machine. On Windows, the folder is located at `C:\users\<username>\AppData\Local`. For Mac and Linux, you can find the folder at the root of your user directory.
4. If you're running Mac or Linux, you also need to delete the `Microsoft.Developer.IdentityService` entry from your operating system's keystore. On the Mac, the keystore is the Gnome Keychain application. In Linux, the application is typically called `Keyring`, but the name might differ depending on your distribution.
5. Restart Storage Explorer and try to sign in again.

#### **macOS: Keychain errors or no sign-in window**

macOS Keychain can sometimes enter a state that causes issues for the Storage Explorer authentication library.

To get Keychain out of this state:

1. Close Storage Explorer.
2. Open Keychain by selecting **Command+Spacebar**, enter **keychain**, and select **Enter**.
3. Select the **login** keychain.
4. Select the **padlock** to lock the keychain. After the process is finished, the **padlock** appears locked. It might take a few seconds, depending on what apps you have open.



5. Open Storage Explorer.
6. You're prompted with a message like "Service hub wants to access the Keychain." Enter your Mac admin account password and select **Always Allow**. Or select **Allow** if **Always Allow** isn't available.
7. Try to sign in.

#### **Default browser doesn't open**

If your default browser doesn't open when you try to sign in, try all of the following techniques:

- Restart Storage Explorer.
- Open your browser manually before you start to sign in.
- Try using **Integrated Sign-In**. For instructions, see [Changing where sign-in happens](#).

#### **Other sign-in issues**

If none of the preceding instructions apply to your sign-in issue or if they fail to resolve your sign-in issue, [open an issue on GitHub](#).

#### **Missing subscriptions and broken tenants**

If you can't retrieve your subscriptions after you successfully sign in, try the following troubleshooting methods:

- Verify that your account has access to the subscriptions you expect. You can verify your access by signing in to the portal for the Azure environment you're trying to use.
- Make sure you've signed in through the correct Azure environment like Azure, Azure China 21Vianet, Azure Germany, Azure US Government, or Custom Environment.
- If you're behind a proxy server, make sure you configured the Storage Explorer proxy correctly.
- Try removing and re-adding the account.
- If there's a "More information" or "Error details" link, check which error messages are being reported for the tenants that are failing. If you aren't sure how to respond to the error messages, [open an issue in GitHub](#).

## Problem interacting with your OS credential store during an AzCopy transfer

If you see this message on Windows, most likely the Windows Credential Manager is full. To make room in the Windows Credential Manager

1. Close Storage Explorer
2. On the **Start** menu, search for **Credential Manager** and open it.
3. Go to **Windows Credentials**.
4. Under **Generic Credentials**, look for entries associated with programs you no longer use and delete them.  
You can also look for entries like `azcopy/aadtoken/<some number>` and delete those.

If the message continues to appear after completing the above steps, or if you encounter this message on platforms other than Windows, then please [open an issue on GitHub](#).

## Can't remove an attached storage account or resource

If you can't remove an attached account or storage resource through the UI, you can manually delete all attached resources by deleting the following folders:

- Windows: `%AppData%/StorageExplorer`
- macOS: `/Users/<your_name>/Library/Application Support/StorageExplorer`
- Linux: `~/.config/StorageExplorer`

Close Storage Explorer before you delete these folders.

### NOTE

If you've ever imported any SSL certificates, back up the contents of the `certs` directory. Later, you can use the backup to reimport your SSL certificates.

## Proxy issues

Storage Explorer supports connecting to Azure Storage resources via a proxy server. If you experience any issues when you connect to Azure via proxy, here are some suggestions.

Storage Explorer only supports basic authentication with proxy servers. Other authentication methods, such as NTLM, aren't supported.

### NOTE

Storage Explorer doesn't support proxy autoconfig files for configuring proxy settings.

## Verify Storage Explorer proxy settings

The Application > Proxy > Proxy configuration setting determines which source Storage Explorer gets the proxy configuration from.

If you select Use environment variables, make sure to set the `HTTPS_PROXY` or `HTTP_PROXY` environment variables. Environment variables are case sensitive, so be sure to set the correct variables. If these variables are undefined or invalid, Storage Explorer won't use a proxy. Restart Storage Explorer after you modify any environment variables.

If you select Use app proxy settings, make sure the in-app proxy settings are correct.

## Steps for diagnosing issues

If you're still experiencing issues, try these troubleshooting methods:

1. If you can connect to the internet without using your proxy, verify that Storage Explorer works without proxy settings enabled. If Storage Explorer connects successfully, there might be an issue with your proxy server. Work with your admin to identify the problems.
2. Verify that other applications that use the proxy server work as expected.
3. Verify that you can connect to the portal for the Azure environment you're trying to use.
4. Verify that you can receive responses from your service endpoints. Enter one of your endpoint URLs into your browser. If you can connect, you should receive an `InvalidParameterValue` or similar XML response.
5. Check whether someone else using Storage Explorer with the same proxy server can connect. If they can, you might have to contact your proxy server admin.

## Tools for diagnosing issues

A networking tool, such as Fiddler, can help you diagnose problems.

1. Configure your networking tool as a proxy server running on the local host. If you have to continue working behind an actual proxy, you might have to configure your networking tool to connect through the proxy.
2. Check the port number used by your networking tool.
3. Configure Storage Explorer proxy settings to use the local host and the networking tool's port number, such as "localhost:8888".

When set correctly, your networking tool will log network requests made by Storage Explorer to management and service endpoints.

If your networking tool doesn't appear to be logging Storage Explorer traffic, try testing your tool with a different application. For example, enter the endpoint URL for one of your storage resources, such as

`https://contoso.blob.core.windows.net/`) in a web browser. You'll receive a response similar to this code sample.

```
<?xml version="1.0" encoding="UTF-8"?>
- <Error>
 <Code>InvalidParameterValue</Code>
 <Message>Value for one of the query parameters specified in the request URI is invalid.
RequestId:57d9d9e5-0001-0008-0143-b28062000000 Time:2017-04-
 10T21:42:17.3863214Z</Message>
 <QueryParameterName>comp</QueryParameterName>
 <QueryParameterValue/>
 <Reason/>
</Error>
```

The response suggests the resource exists, even though you can't access it.

If your networking tool only shows traffic from other applications, you might need to adjust the proxy settings in Storage Explorer. Otherwise, you might need to adjust your tool's settings.

## Contact proxy server admin

If your proxy settings are correct, you might have to contact your proxy server admin to:

- Make sure your proxy doesn't block traffic to Azure management or resource endpoints.
- Verify the authentication protocol used by your proxy server. Storage Explorer only supports basic authentication protocols. Storage Explorer doesn't support NTLM proxies.

## "Unable to Retrieve Children" error message

If you're connected to Azure through a proxy, verify that your proxy settings are correct.

If the owner of a subscription or account has granted you access to a resource, verify that you have read or list permissions for that resource.

## Connection string doesn't have complete configuration settings

If you receive this error message, it's possible that you don't have the necessary permissions to obtain the keys for your storage account. To confirm that this is the case, go to the portal and locate your storage account. Right-click the node for your storage account and select **Open in Portal**. Then, go to the **Access Keys** pane. If you don't have permissions to view keys, you'll see a "You don't have access" message. To work around this issue, you can either obtain the account key from someone else and attach through the name and key or you can ask someone for a shared access signature to the storage account and use it to attach the storage account.

If you do see the account keys, file an issue in GitHub so that we can help you resolve the issue.

## "Error occurred while adding new connection: TypeError: Cannot read property 'version' of undefined"

If you receive this error message when you try to add a custom connection, the connection data that's stored in the local credential manager might be corrupted. To work around this issue, try deleting your corrupted local connections, and then re-add them:

1. Start Storage Explorer. From the menu, go to **Help > Toggle Developer Tools**.
2. In the opened window, on the **Application** tab, go to **Local Storage > file://** on the left side.
3. Depending on the type of connection you're having an issue with, look for its key. Then copy its value into a text editor. The value is an array of your custom connection names, such as:
  - Storage accounts
    - `StorageExplorer_CustomConnections_Accounts_v1`
  - Blob containers
    - `StorageExplorer_CustomConnections_Blobs_v1`
    - `StorageExplorer_CustomConnections_Blobs_v2`
  - File shares
    - `StorageExplorer_CustomConnections_Files_v1`
  - Queues
    - `StorageExplorer_CustomConnections_Queue_v1`
  - Tables
    - `StorageExplorer_CustomConnections_Tables_v1`
4. After you save your current connection names, set the value in **Developer Tools** to `[]`.

To preserve the connections that aren't corrupted, use the following steps to locate the corrupted connections. If you don't mind losing all existing connections, skip these steps and follow the platform-specific instructions to clear your connection data.

1. From a text editor, re-add each connection name to **Developer Tools**. Then check whether the connection is still working.
2. If a connection is working correctly, it's not corrupted and you can safely leave it there. If a connection isn't working, remove its value from **Developer Tools**, and record it so that you can add it back later.
3. Repeat until you've examined all your connections.

After you go through all your connections, for all connection names that aren't added back, you must clear their corrupted data, if there is any. Then add them back by using the standard steps in Storage Explorer.

- [Windows](#)
- [macOS](#)
- [Linux](#)

1. On the **Start** menu, search for **Credential Manager** and open it.
2. Go to **Windows Credentials**.
3. Under **Generic Credentials**, look for entries that have the `<connection_type_key>/<corrupted_connection_name>` key. An example is `StorageExplorer_CustomConnections_Accounts_v1/account1`.
4. Delete these entries and re-add the connections.

If you still encounter this error after you run these steps, or if you want to share what you suspect has corrupted the connections, [open an issue](#) on our GitHub page.

## Issues with a shared access signature URL

If you connect to a service through a shared access signature URL and experience an error:

- Verify that the URL provides the necessary permissions to read or list resources.
- Verify that the URL hasn't expired.
- If the shared access signature URL is based on an access policy, verify that the access policy hasn't been revoked.

If you accidentally attached by using an invalid shared access signature URL and now can't detach, follow these steps:

1. When you're running Storage Explorer, select **F12** to open the **Developer Tools** window.
2. On the **Application** tab, select **Local Storage** > `file://` on the left side.
3. Find the key associated with the service type of the problematic shared access signature URI. For example, if the bad shared access signature URI is for a blob container, look for the key named `StorageExplorer_AddStorageServiceSAS_v1_blob`.
4. The value of the key should be a JSON array. Find the object associated with the bad URI, and delete it.
5. Select **Ctrl+R** to reload Storage Explorer.

## Linux dependencies

### Snap

Storage Explorer 1.10.0 and later is available as a snap from the Snap Store. The Storage Explorer snap installs all its dependencies automatically. It's updated when a new version of the snap is available. Installing the Storage Explorer snap is the recommended method of installation.

Storage Explorer requires the use of a password manager, which you might need to connect manually before Storage Explorer will work correctly. You can connect Storage Explorer to your system's password manager by running the following command:

```
snap connect storage-explorer:password-manager-service :password-manager-service
```

### .tar.gz file

You can also download the application as a *.tar.gz* file, but you'll have to install dependencies manually.

Storage Explorer as provided in the *.tar.gz* download is supported for the following versions of Ubuntu only.

Storage Explorer might work on other Linux distributions, but they aren't officially supported.

- Ubuntu 20.04 x64
- Ubuntu 18.04 x64
- Ubuntu 16.04 x64

Storage Explorer requires the .NET 6 runtime to be installed on your system. The ASP.NET runtime is **not** required.

#### NOTE

Older versions of Storage Explorer may require a different version of .NET or .NET Core. Refer to release notes or in app error messages to help determine the required version.

- [Ubuntu 22.04](#)
- [Ubuntu 20.04](#)
- [Ubuntu 18.04](#)

1. Download the Storage Explorer *.tar.gz* file.

2. Install the [.NET 6 runtime](#)

Many libraries needed by Storage Explorer come preinstalled with Canonical's standard installations of Ubuntu.

Custom environments might be missing some of these libraries. If you have issues launching Storage Explorer, make sure the following packages are installed on your system:

- iproute2
- libasound2
- libatm1
- libgconf-2-4
- libnspr4
- libnss3
- libpulse0
- libsecret-1-0
- libx11-xcb1
- libxss1
- libxtables11
- libxtst6
- xdg-utils

### Patch Storage Explorer for newer versions of .NET Core

For Storage Explorer 1.7.0 or earlier, you might have to patch the version of .NET Core used by Storage Explorer:

1. Download version 1.5.43 of StreamJsonRpc [from NuGet](#).1. Look for the **Download package** link on the right side of the page.
2. After you download the package, change its file extension from *.nupkg* to *.zip*.

3. Unzip the package.
4. Open the `streamjsonrpc.1.5.43/lib/netstandard1.1/` folder.
5. Copy `StreamJsonRpc.dll` to the following locations in the Storage Explorer folder:
  - `StorageExplorer/resources/app/ServiceHub/Services/Microsoft.Developer.IdentityService/`
  - `StorageExplorer/resources/app/ServiceHub/Hosts/ServiceHub.Host.Core.CLR.x64/`

## Open In Explorer button in the Azure portal doesn't work

If the **Open In Explorer** button in the Azure portal doesn't work, make sure you're using a compatible browser. The following browsers were tested for compatibility:

- Microsoft Edge
- Mozilla Firefox
- Google Chrome
- Microsoft Internet Explorer

## Gather logs

When you report an issue to GitHub, you might be asked to gather certain logs to help diagnose your issue.

### Storage Explorer logs

Starting with version 1.16.0, Storage Explorer logs various things to its own application logs. You can easily get to these logs by selecting **Help > Open Logs Directory**. By default, Storage Explorer logs at a low level of verbosity. To change the verbosity level, add an environment variable with the name of `STG_EX_LOG_LEVEL`, and any of the following values:

- `silent`
- `critical`
- `error`
- `warning`
- `info` (default level)
- `verbose`
- `debug`

Logs are split into folders for each session of Storage Explorer that you run. For whatever log files you need to share, place them in a zip archive, with files from different sessions in different folders.

### Authentication logs

For issues related to sign-in or Storage Explorer's authentication library, you'll most likely need to gather authentication logs. Authentication logs are stored at:

- Windows: `C:\Users\<your username>\AppData\Local\Temp\servicehub\logs`
- macOS and Linux: `~/ServiceHub/logs`

Generally, you can follow these steps to gather the logs:

1. Go to **Settings** (the **gear** symbol on the left) > **Application** > **Sign-in**. Select **Verbose Authentication Logging**. If Storage Explorer fails to start because of an issue with its authentication library, this will be done for you.
2. Close Storage Explorer.
3. Optional/recommended: Clear out existing logs from the `logs` folder. This step reduces the amount of information you have to send us.

4. Open Storage Explorer and reproduce your issue.
5. Close Storage Explorer.
6. Zip the contents of the `/logs` folder.

## AzCopy logs

If you're having trouble transferring data, you might need to get the AzCopy logs. AzCopy logs can be found easily via two different methods:

- For failed transfers still in the Activity Log, select [Go to AzCopy Log File](#).
- For transfers that failed in the past, go to the AzCopy logs folder. This folder can be found at:
  - Windows: `C:\Users\<your username>\.azcopy`
  - macOS and Linux: `~/.azcopy`

## Network logs

For some issues, you'll need to provide logs of the network calls made by Storage Explorer. On Windows, you can do this step by using Fiddler.

### NOTE

Fiddler traces might contain passwords you entered or sent in your browser during the gathering of the trace. Make sure to read the instructions on how to sanitize a Fiddler trace. Don't upload Fiddler traces to GitHub. You'll be told where you can securely send your Fiddler trace.

### Part 1: Install and configure Fiddler

1. Install Fiddler.
2. Start Fiddler.
3. Go to **Tools > Options**.
4. Select the **HTTPS** tab.
5. Make sure **Capture CONNECTs** and **Decrypt HTTPS traffic** are selected.
6. Select **Actions**.
7. Select **Trust Root Certificate** and then select **Yes** in the next dialog.
8. Select **Actions** again.
9. Select **Export Root Certificate to Desktop**.
10. Go to your desktop, find the `FiddlerRoot.cer` file, and double-click it.
11. Go to the **Details** tab.
12. Select **Copy to File**.
13. In the export wizard, choose the following options:
  - Base-64 encoded X.509.
  - For file name, browse to `C:\Users\<your user dir>\AppData\Roaming\StorageExplorer\certs`. Then you can save it as any file name.
14. Close the certificate window.
15. Start Storage Explorer.
16. Go to **Edit > Configure Proxy**.

17. In the dialog, select **Use app proxy settings**. Set the URL to `http://localhost` and the port to **8888**.

18. Select **OK**.

19. Restart Storage Explorer.

20. You should start seeing network calls from a `storageexplorer:` process show up in Fiddler.

#### Part 2: Reproduce the issue

1. Close all apps other than Fiddler.
2. Clear the Fiddler log by using the **X** in the top left, near the **View** menu.
3. Optional/recommended: Let Fiddler set for a few minutes. If you see network calls appear that aren't related to Storage Explorer, right-click them and select **Filter Now > Hide (process name)**.
4. Start Storage Explorer.
5. Reproduce the issue.
6. Select **File > Save > All Sessions**. Save it somewhere you won't forget.
7. Close Fiddler and Storage Explorer.

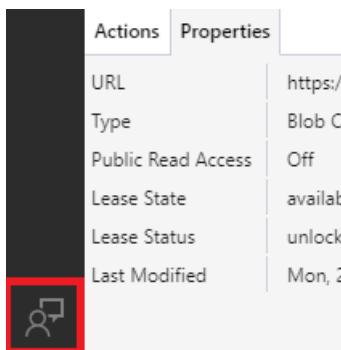
#### Part 3: Sanitize the Fiddler trace

1. Double-click the Fiddler trace (.saz file).
2. Select **Ctrl+F**.
3. In the dialog that appears, make sure the following options are set: **Search = Requests and responses** and **Examine = Headers and bodies**.
4. Search for any passwords you used while you collected the Fiddler trace and any entries that are highlighted. Right-click and select **Remove > Selected sessions**.
5. If you definitely entered passwords into your browser while you collected the trace but you don't find any entries when you use **Ctrl+F**, you don't want to change your passwords, or if the passwords you used are used for other accounts, skip sending us the .saz file.
6. Save the trace again with a new name.
7. Optional: Delete the original trace.

## Next steps

If none of these solutions work for you, you can:

- [Create a support ticket](#).
- [Open an issue on GitHub](#) by selecting the **Report issue to GitHub** button in the lower-left corner.



# Connect an emulator to Storage Explorer

8/22/2022 • 2 minutes to read • [Edit Online](#)

Storage Explorer can be connected to an Azure Storage emulator to aid in development. Emulators provide a free local environment for testing your code and applications. And once you're satisfied with how your application is working locally, you can switch to using a real Azure Storage account in the cloud.

## Supported emulators

Storage Explorer officially supports and recommends using [Azurite](#). Azurite is a cross platform, open-source emulator made by Microsoft. It supports blobs, queues, and tables. However, any emulator that functions similarly to Azurite will also likely work with Storage Explorer. Also note that Storage Explorer doesn't ship with an emulator. You'll need to download an emulator on your own.

## How to connect to an emulator

Before you can connect Storage Explorer to an emulator, you must first start the emulator. Storage Explorer doesn't start your emulator for you. If you attempt to access an emulator storage account before starting your emulator, you'll receive a message reminding you to start your emulator. If you attempt to access an emulator container, queue, or table, before starting your emulator you'll receive an error message.

Assuming you have started your emulator and it is:

- Running locally
- Configured to listen on the typical default ports of:
  - Blobs: `10000`
  - Queues: `10001`
  - Tables: `10002`
- Configured to use HTTP

Then you can quickly access your emulator resources by navigating in the resource tree view to **Local & Attached > Storage Accounts > Emulator (Default Ports)**.

If any of the above aren't true, then you'll need to manually add a connection to your emulator.

## Manually connect to an emulator

### Local emulator

If you need to manually connect to an emulator that is running locally, open the Connect dialog (plug icon in the left side) and choose **Local storage emulator**. Then fill out all required fields. Make sure to set the ports for each service type to their appropriate values. If your emulator is configured to use HTTPS, also make sure to check the checkbox for HTTPS. After you finish connecting, you can find the storage account node for your emulator under **Local & Attached > Storage Accounts**.

### Remote emulator

#### **NOTE**

Although Storage Explorer supports connecting to a remote emulator, it is not recommended. Certain remote emulator configurations may make it difficult for Storage Explorer to understand that an emulator is being connected to, which may affect some features. For best results, configure your emulator to use the default development account name and key, and then use a name and key connection string to connect.

If you need to manually connect to an emulator running on a different machine, then you will need to construct a connection string that details how to connect to your emulator. The connection string will likely need to explicitly define the endpoint for each service type.

Once you have a connection string, open the Connect dialog (plug icon in the left side) and choose **Storage account or service**. Then select the **Connection string** option, proceed to the next page, and use your connection string to complete the connection process. You can then find the storage account node for your emulator under **Local & Attached > Storage Accounts**.

## Next steps

- [Troubleshoot issues](#)

# Azure Storage Explorer command-line options

8/22/2022 • 2 minutes to read • [Edit Online](#)

Microsoft Azure Storage Explorer has a set of command-line options that can be added when starting the application. Most of the command-line options are for debugging or troubleshooting purposes.

## Command-line options

OPTION	DESCRIPTION
<code>--debug</code> / <code>--prod</code>	Start the application in debug or production mode. In debug mode, the local attachment data will be stored in the application's local storage and it won't be encrypted. Hidden properties will be displayed in the Properties panel for selected resource nodes. Log verbosity level will be set to print debug messages revealing Storage Explorer's internal setup logic. The default value is <code>--prod</code> .
<code>--lang</code>	Start the application with a given language. For example, <code>--lang="zh-Hans"</code> .
<code>--disable-gpu</code>	Start the application without GPU acceleration.
<code>--auto-open-dev-tools</code>	Let the application open the developer tools window as soon as the browser window shows. This option is useful when you want to hit a break point at a line in the start-up code of the browser window.
<code>--verbosity</code>	Set the verbosity level of Storage Explorer logging. Supported verbosity levels include <code>debug</code> , <code>verbose</code> , <code>info</code> , <code>warn</code> , <code>error</code> , and <code>silent</code> . For example, <code>--verbosity=verbose</code> . When running in production mode, the default verbosity level is <code>info</code> . When running in debug mode, the log verbosity level will always be <code>debug</code> .
<code>--log-dir</code>	Set the directory to save log files. For example, <code>--log-dir=path_to_a_directory</code> .
<code>--ignore-certificate-errors</code>	Tell Storage Explorer to ignore certificate errors. This flag can be useful when you need to work in a trusted proxy environment with non-public Certificate Authority. We recommend you to <a href="#">use system proxy (preview)</a> in such proxy environments and only set this flag if the system proxy doesn't work.

An example of starting Storage Explorer with custom command-line options

```
./MicrosoftAzureStorageExplorer --lang=en --auto-open-dev-tools
```

**NOTE**

These command line options may change in new Storage Explorer versions.

## When to use command-line options

Some command-line options can be used to customize Storage Explorer. For those options that have corresponding user settings, such as `--lang`. We recommend using the user settings instead of using the command-line option.

The other command-line options can be useful for debugging and troubleshooting. If you run into a problem in Storage Explorer, reproducing the problem in debug mode can help us get more detailed information to investigate.

# Azure Storage Explorer direct link

8/22/2022 • 2 minutes to read • [Edit Online](#)

On Windows and macOS, Storage Explorer supports URLs with the `storageexplorer://` protocol. These URLs are referred to as direct links. A direct link points to an Azure Storage resource in Storage Explorer. Following a direct link will open Storage Explorer and navigate to the resource it points to. This article describes how direct links work and how you can use them.

## How direct links work

Storage Explorer uses the tree view to visualize resources in Azure. A direct link contains the hierarchical information for the linked resource node in the tree. When a direct link is followed, Storage Explorer opens and receives the parameters in the direct link. Storage Explorer then uses these parameters to navigate to the linked resource in the tree view.

### IMPORTANT

You must be signed in and have the necessary permissions to access the linked resource for a direct link to work.

## Parameters

A Storage Explorer direct link always starts with protocol `storageexplorer://`. The following table explains each of the possible parameters in a direct link.

PARAMETER	DESCRIPTION
<code>v</code>	Version of the direct link protocol.
<code>accountid</code>	The Azure Resource Manager resource ID of the storage account for the linked resource. If the linked resource is a storage account, this ID will be the Azure Resource Manager resource ID of that storage account. Otherwise, this ID will be the Azure Resource Manager resource ID of the storage account the linked resource belongs to.
<code>resourcetype</code>	Optional. Only used when the linked resource is a blob container, a file share, a queue, or a table. Must be either one of "Azure.BlobContainer", "Azure.FileShare", "Azure.Queue", "Azure.FileShare".
<code>resourcename</code>	Optional. Only used when the linked resource is a blob container, a file share, a queue, or a table. The name of the linked resource.

Here is an example direct link to a blob container.

`storageexplorer://v=1&accountid=/subscriptions/the_subscription_id/resourceGroups/the_resource_group_name/providers/Microsoft.Storage/storageAccounts/the_stora`

## Get a direct link from Storage Explorer

You can use Storage Explorer to get a direct link for a resource. Open the context menu of the node for the resource in the tree view. Then use the "Copy Direct Link" action to copy its direct link to the clipboard.

## Direct link limitations

Direct links are only supported for resources under subscription nodes. Additionally, only the following resource types are supported:

- Storage Accounts
- Blob Containers
- Queues
- File Shares
- Tables

# Azure Storage Explorer security guide

8/22/2022 • 4 minutes to read • [Edit Online](#)

Microsoft Azure Storage Explorer enables you to easily work with Azure Storage data safely and securely on Windows, macOS, and Linux. By following these guidelines, you can ensure your data stays protected.

## General

- **Always use the latest version of Storage Explorer.** Storage Explorer releases may contain security updates. Staying up to date helps ensure general security.
- **Only connect to resources you trust.** Data that you download from untrusted sources could be malicious, and uploading data to an untrusted source may result in lost or stolen data.
- **Use HTTPS whenever possible.** Storage Explorer uses HTTPS by default. Some scenarios allow you to use HTTP, but HTTP should be used only as a last resort.
- **Ensure only the needed permissions are given to the people who need them.** Avoid being overly permissive when granting anyone access to your resources.
- **Use caution when executing critical operations.** Certain operations, such as delete and overwrite, are irreversible and may cause data loss. Make sure you're working with the correct resources before executing these operations.

## Choosing the right authentication method

Storage Explorer provides various ways to access your Azure Storage resources. Whatever method you choose, here are our recommendations.

### Azure AD authentication

The easiest and most secure way to access your Azure Storage resources is to sign in with your Azure account. Signing in uses Azure AD authentication, which allows you to:

- Give access to specific users and groups.
- Revoke access to specific users and groups at any time.
- Enforce access conditions, such as requiring multi-factor authentication.

We recommend using Azure AD authentication whenever possible.

This section describes the two Azure AD-based technologies that can be used to secure your storage resources.

#### Azure role-based access control (Azure RBAC)

[Azure role-based access control \(Azure RBAC\)](#) give you fine-grained access control over your Azure resources. Azure roles and permissions can be managed from the Azure portal.

Storage Explorer supports Azure RBAC access to Storage Accounts, Blobs, and Queues. If you need access to File Shares or Tables, you'll need to assign Azure roles that grant permission to list storage account keys.

#### Access control lists (ACLs)

[Access control lists \(ACLs\)](#) let you control file and folder level access in ADLS Gen2 blob containers. You can manage your ACLs using Storage Explorer.

#### Shared access signatures (SAS)

If you can't use Azure AD authentication, we recommend using shared access signatures. With shared access signatures, you can:

- Provide anonymous limited access to secure resources.
- Revoke a SAS immediately if generated from a shared access policy (SAP).

However, with shared access signatures, you can't:

- Restrict who can use a SAS. A valid SAS can be used by anyone who has it.
- Revoke a SAS if not generated from a shared access policy (SAP).

When using SAS in Storage Explorer, we recommend the following guidelines:

- **Limit the distribution of SAS tokens and URIs.** Only distribute SAS tokens and URIs to trusted individuals. Limiting SAS distribution reduces the chance a SAS could be misused.
- **Only use SAS tokens and URIs from entities you trust.**
- **Use shared access policies (SAP) when generating SAS tokens and URIs if possible.** A SAS based on a shared access policy is more secure than a bare SAS, because the SAS can be revoked by deleting the SAP.
- **Generate tokens with minimal resource access and permissions.** Minimal permissions limit the potential damage that could be done if a SAS is misused.
- **Generate tokens that are only valid for as long as necessary.** A short lifespan is especially important for bare SAS, because there's no way to revoke them once generated.

#### IMPORTANT

When sharing SAS tokens and URIs for troubleshooting purposes, such as in service requests or bug reports, always redact at least the signature portion of the SAS.

## Storage account keys

Storage account keys grant unrestricted access to the services and resources within a storage account. For this reason, we recommend limiting the use of keys to access resources in Storage Explorer. Use Azure RBAC features or SAS to provide access instead.

Some Azure roles grant permission to retrieve storage account keys. Individuals with these roles can effectively circumvent permissions granted or denied by Azure RBAC. We recommend not granting this permission unless it's necessary.

Storage Explorer will attempt to use storage account keys, if available, to authenticate requests. You can disable this feature in Settings (**Services > Storage Accounts > Disable Usage of Keys**). Some features don't support Azure RBAC, such as working with Classic storage accounts. Such features still require keys and are not affected by this setting.

If you must use keys to access your storage resources, we recommend the following guidelines:

- **Don't share your account keys with anyone.**
- **Treat your storage account keys like passwords.** If you must make your keys accessible, use secure storage solutions such as [Azure Key Vault](#).

#### NOTE

If you believe a storage account key has been shared or distributed by mistake, you can generate new keys for your storage account from the Azure portal.

## Public access to blob containers

Storage Explorer allows you to modify the access level of your Azure Blob Storage containers. Non-private blob containers allow anyone anonymous read access to data in those containers.

When enabling public access for a blob container, we recommend the following guidelines:

- **Don't enable public access to a blob container that may contain any potentially sensitive data.**  
Make sure your blob container is free of all private data.
- **Don't upload any potentially sensitive data to a blob container with Blob or Container access.**

## Next steps

- [Security recommendations](#)

# Azure Storage Explorer soft delete guide

8/22/2022 • 3 minutes to read • [Edit Online](#)

Soft delete helps mitigate the impact of accidentally deleting critical data. This guide will help you understand how you can take advantage of this feature in Storage Explorer. Before continuing, it's recommended you read more about [blob soft delete](#).

## Configuring delete retention policy

You can configure the delete retention policy for each storage account in Storage Explorer. Open the context menu for any "Blob Containers" node under the storage account and choose **Configure Soft Delete Policy....**

Setting a new policy may take up to 30 seconds for it to take effect. Deleting data without waiting for the new policy to take effect may result in unexpected behavior. Storage Explorer waits 30 seconds before reporting a successfully configured policy in the Activity Log.

## Soft delete with hierarchical namespace enabled

The soft delete feature has fundamental differences between blob containers with or without hierarchical namespace (HNS) enabled.

HNS enabled blob containers have real directories. Those directories can also be soft-deleted. When a real directory is soft-deleted, all the active blobs or directories under it will become inaccessible. These blobs and directories will be recovered when the directory is undeleted and discarded when the directory expires. Blobs or directories under it that have already been soft-deleted will be kept as is.

Non-HNS enabled blob containers don't allow soft-deleted blobs and active blobs with the same name to coexist. Uploading a blob with the same name as a soft-deleted blob will cause the soft-deleted blob to become a snapshot of the new one. In an HNS enabled blob container, doing the same thing will result in the soft-deleted blob coexisting with the new one. HNS enabled blob containers also allow the coexistence of multiple soft-deleted blobs with the same name.

Each soft-deleted blob or directory in an HNS enabled blob container has a `DeletionID` property. This property distinguishes blobs or directories with the same name. Soft-deleted blobs in non-HNS enabled blob containers don't have a `DeletionID` property.

Non-HNS enabled blob containers also support "blob versioning". If blob versioning is enabled, the behavior of certain operations, such as delete, changes. For more information, see [Azure Storage Explorer blob versioning guide](#).

## View soft-deleted blobs

In Blob Explorer, soft-deleted blobs are shown under certain view contexts.

For blob containers without HNS enabled, choose the "Active blobs and soft deleted blobs" or the "All blobs and blobs without current version" view context to view soft-deleted blobs.

For blob containers with HNS enabled, choose the "Active and soft deleted blobs" or the "Deleted only" view context to view soft-deleted blobs and directories.

## Delete blobs

When deleting blobs or directories, Storage Explorer checks the storage account's current delete retention policy. A confirmation dialog then informs you what will happen if you continue with the delete operation. If soft delete is disabled, you can enable it from the confirmation dialog by selecting the **Enable Soft Delete** button.

#### WARNING

Storage Explorer may not see a new delete retention policy if you just saved it. It is highly recommended that you wait at least 30 seconds for the new policy to take effect before deleting data.

## Undelete blobs

Storage Explorer can undelete soft-deleted blobs recursively and in batches.

To undelete blobs, select the soft-deleted blobs you want to undelete and use the **Undelete → Undelete Selected** from the toolbar or the context menu.

You can also undelete blobs recursively under a directory. If an active directory is included in the selection, Storage Explorer will undelete all the soft-deleted blobs or directories in it.

In HNS enabled blob containers, undeleting a blob will fail if an active blob with the same name already exists.

#### NOTE

Soft-deleted snapshots can only be undeleted by undeleting the base blob. There is no way to undelete individual snapshots.

## Undelete blobs by date range

Storage Explorer also lets you undelete blobs based on their deletion time.

To undelete blobs by date range, select the soft-deleted blobs you want to undelete and use the **Undelete → Undelete by Date...** from the toolbar or the context menu.

**Undelete by Date...** works exactly the same as **Undelete Selected**, except that it will filter out blobs or directories whose deletion time is out of the range you specify.

## See Also

- [Azure Storage Explorer blob versioning guide](#)
- [Soft delete for blobs](#)

# Azure Storage Explorer blob versioning guide

8/22/2022 • 3 minutes to read • [Edit Online](#)

Microsoft Azure Storage Explorer provides easy access and management of blob versions. This guide will help you understand how blob versioning works in Storage Explorer. Before continuing, it's recommended you read more about [blob versioning](#).

## Terminology

This section provides some definitions to help understand their usage in this article.

- Soft delete: An alternative automatic data protection feature. You can learn more about soft delete [here](#).
- Active blob: A blob or blob version is created in active state. You can only operate on blobs or blob versions in active state.
- Soft-deleted blob: A blob or blob version marked as soft-deleted. Soft-deleted blobs are only kept for its retention period.
- Blob version: A blob created with blob versioning enabled. Each blob version is associated with a version ID.
- Current version: A blob version marked as the current version.
- Previous version: A blob version that isn't the current version.
- Non-version blob: A blob created with blob versioning disabled. A non-version blob doesn't have a version ID.

## View blob versions

Storage Explorer supports four different views for viewing blobs.

VIEW	ACTIVE NON-VERSION BLOBS	SOFT-DELETED NON-VERSION BLOBS	BLOB VERSIONS
Active blobs	Yes	No	Current version only
Active blobs and soft-deleted blobs	Yes	Yes	Current version only
Active blobs and blobs without current version	Yes	No	Current version or latest active version
All blobs and blobs without current version	Yes	Yes	Current version or latest version

### Active blobs

In this view, Storage Explorer displays:

- Active non-version blobs
- Current versions

### Active blobs and soft-deleted blobs

In this view, Storage Explorer displays:

- Active non-version blobs

- Soft-deleted non-version blobs
- Current versions.

### Active blobs and blobs without current version

In this view, Storage Explorer displays:

- Active non-version blobs
- Current versions
- Latest active previous versions.

For blobs that don't have a current version but have an active previous version, Storage Explorer displays their latest active previous version as a representation of that blob.

### All blobs and blobs without current version

In this view, Storage Explorer displays:

- Active non-version blobs
- Soft-deleted non-version blobs
- Current versions
- Latest previous versions.

For blobs that don't have a current version, Storage Explorer displays their latest previous version as a representation of that blob.

#### NOTE

Due to service limitation, Storage Explorer needs some additional processing to get a hierarchical view of your virtual directories when listing blob versions. It will take longer to list blobs in the following views:

- Active blobs and blobs without current version
- All blobs and blobs without current version

## Manage blob versions

### View versions of a blob

Storage Explorer provides a **Manage Versions** command to view all the versions of a blob. To view a blob's versions, select the blob you want to view versions for and select **Manage History** → **Manage Versions** from either the toolbar or the context menu.

### Download blob versions

To download one or more blob versions, select the blob versions you want to download and select **Download** from the toolbar or the context menu.

If you're downloading multiple versions of a blob, the downloaded files will have their version IDs at the beginning of their file names.

### Delete blob versions

To delete one or more blob versions, select the blob versions you want to delete and select **Delete** from the toolbar or the context menu.

Blob versions are subject to your soft-delete policy. If soft-delete is enabled, blob versions will be soft-deleted. One special case is deleting a current version. Deleting a current version will automatically make it become an active previous version instead.

### Promote blob version

You can restore the contents of a blob by promoting a previous version to become the current version. Select the blob version you want to promote and select **Promote Version** from the toolbar or the context menu.

Non-version blobs will be overwritten by the promoted blob version. Make sure you no longer need that data or back up the data yourself before confirming the operation. Current versions automatically become previous versions, so Storage Explorer won't prompt for confirmation.

### **Undelete blob version**

Blob versions can't be undeleted individually. They must be undeleted all at once. To undelete all blob versions of a blob, select any one of the blob's versions and select **Undelete Selected** from the toolbar or the context menu.

### **Change access tier of blob versions**

Each blob version has its own access tier. To change access tier of blob versions, select the blob versions you want to change access tier and select **Change Access Tier...** from the context menu.

## See Also

- [Blob versioning](#)
- [Soft delete for blobs](#)
- [Azure Storage Explorer soft delete guide](#)

# Manage Azure Blob Storage resources with Storage Explorer

8/22/2022 • 7 minutes to read • [Edit Online](#)

## Overview

[Azure Blob Storage](#) is a service for storing large amounts of unstructured data, such as text or binary data, that can be accessed from anywhere in the world via HTTP or HTTPS. You can use Blob storage to expose data publicly to the world, or to store application data privately. In this article, you'll learn how to use Storage Explorer to work with blob containers and blobs.

## Prerequisites

To complete the steps in this article, you'll need the following:

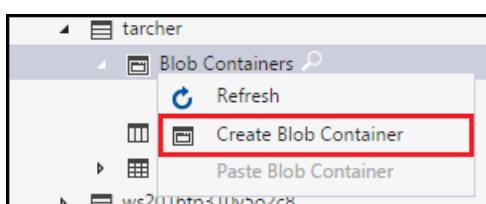
- [Download and install Storage Explorer](#)
- [Connect to an Azure storage account or service](#)

## Create a blob container

All blobs must reside in a blob container, which is simply a logical grouping of blobs. An account can contain an unlimited number of containers, and each container can store an unlimited number of blobs.

The following steps illustrate how to create a blob container within Storage Explorer.

1. Open Storage Explorer.
2. In the left pane, expand the storage account within which you wish to create the blob container.
3. Right-click **Blob Containers**, and - from the context menu - select **Create Blob Container**.



4. A text box will appear below the **Blob Containers** folder. Enter the name for your blob container. See [Create a container](#) for information on rules and restrictions on naming blob containers.



5. Press **Enter** when done to create the blob container, or **Esc** to cancel. Once the blob container has been successfully created, it will be displayed under the **Blob Containers** folder for the selected storage account.

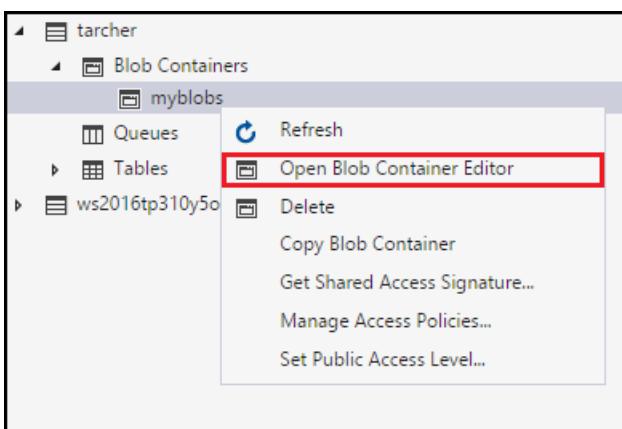


## View a blob container's contents

Blob containers contain blobs and folders (that can also contain blobs).

The following steps illustrate how to view the contents of a blob container within Storage Explorer:

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the blob container you wish to view.
3. Expand the storage account's **Blob Containers**.
4. Right-click the blob container you wish to view, and - from the context menu - select **Open Blob Container Editor**. You can also double-click the blob container you wish to view.



5. The main pane will display the blob container's contents.

A screenshot of the Storage Explorer main pane. The left pane shows 'myblobs' selected. The right pane displays a table of blob contents:

Name	Last Modified	Blob Type	Content Type	Size
reachin.png	Tue, 17 May 2016 20:21:44 GMT	Block Blob	image/png	437.8 KB

Showing 1 to 1 of 1 loaded items.

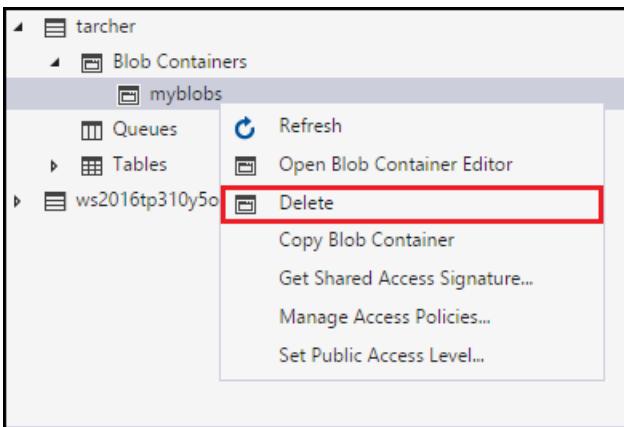
## Delete a blob container

Blob containers can be easily created and deleted as needed. (To see how to delete individual blobs, refer to the section, [Managing blobs in a blob container](#).)

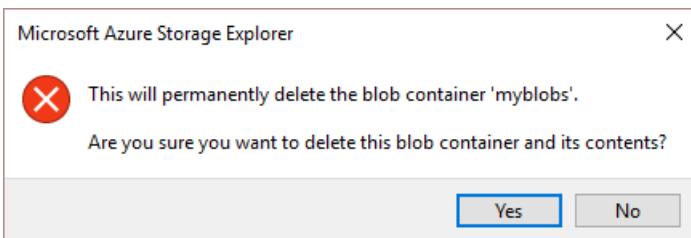
The following steps illustrate how to delete a blob container within Storage Explorer:

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the blob container you wish to view.

3. Expand the storage account's **Blob Containers**.
4. Right-click the blob container you wish to delete, and - from the context menu - select **Delete**. You can also press **Delete** to delete the currently selected blob container.



5. Select **Yes** to the confirmation dialog.

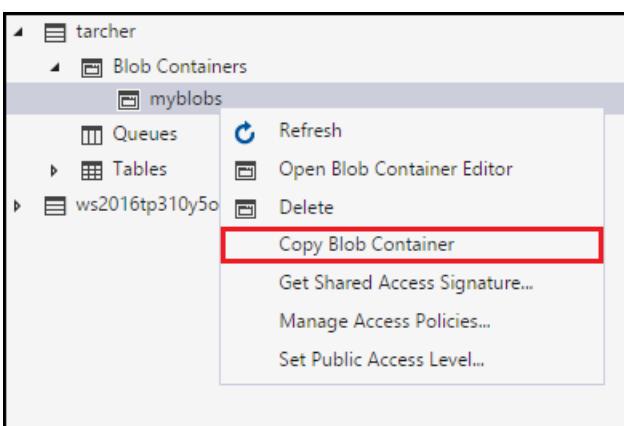


## Copy a blob container

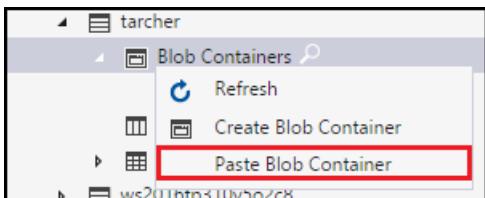
Storage Explorer enables you to copy a blob container to the clipboard, and then paste that blob container into another storage account. (To see how to copy individual blobs, refer to the section, [Managing blobs in a blob container](#).)

The following steps illustrate how to copy a blob container from one storage account to another.

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the blob container you wish to copy.
3. Expand the storage account's **Blob Containers**.
4. Right-click the blob container you wish to copy, and - from the context menu - select **Copy Blob Container**.



5. Right-click the desired "target" storage account into which you want to paste the blob container, and - from the context menu - select **Paste Blob Container**.

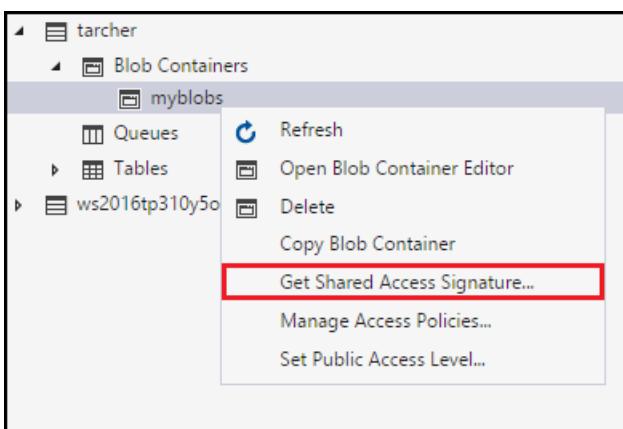


## Get the SAS for a blob container

A [shared access signature \(SAS\)](#) provides delegated access to resources in your storage account. This means that you can grant a client limited permissions to objects in your storage account for a specified period of time and with a specified set of permissions, without having to share your account access keys.

The following steps illustrate how to create a SAS for a blob container:

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the blob container for which you wish to get a SAS.
3. Expand the storage account's **Blob Containers**.
4. Right-click the desired blob container, and - from the context menu - select **Get Shared Access Signature**.



5. In the **Shared Access Signature** dialog, specify the policy, start and expiration dates, time zone, and access levels you want for the resource.

### Shared Access Signature

Access policy:

Start time:

Expiry time:

Time zone:  
 Local  
 UTC

Permissions:

Read  
 Write  
 Delete  
 List

6. When you're finished specifying the SAS options, select **Create**.
7. A second **Shared Access Signature** dialog will then display that lists the blob container along with the URL and QueryStrings you can use to access the storage resource. Select **Copy** next to the URL you wish to copy to the clipboard.

### Shared Access Signature

Container:

URL:

Query string:

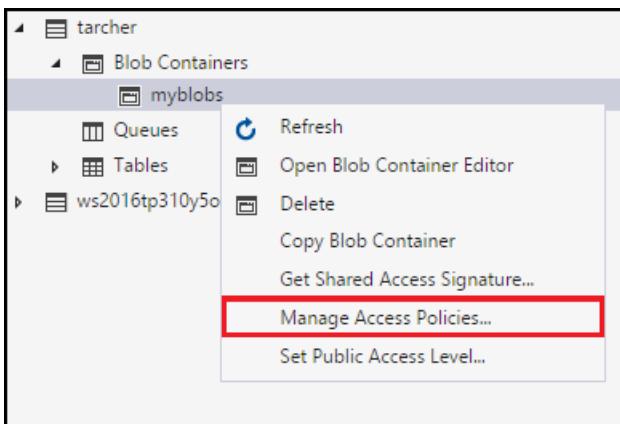
8. When done, select **Close**.

## Manage Access Policies for a blob container

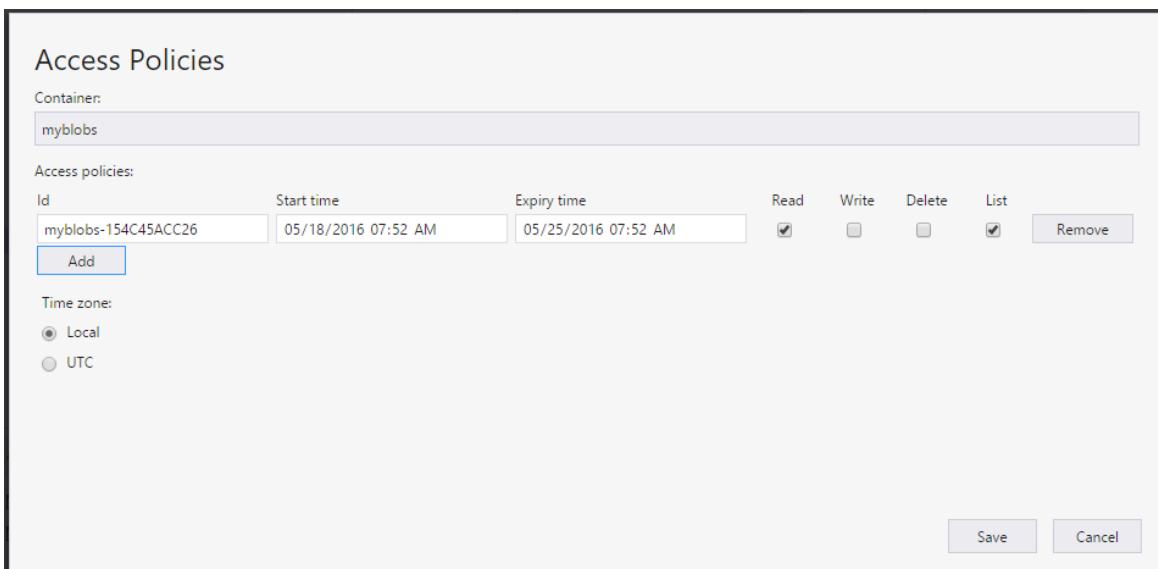
The following steps illustrate how to manage (add and remove) access policies for a blob container:

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the blob container whose access policies you wish to manage.
3. Expand the storage account's **Blob Containers**.

4. Select the desired blob container, and - from the context menu - select **Manage Access Policies**.



5. The **Access Policies** dialog will list any access policies already created for the selected blob container.



6. Follow these steps depending on the access policy management task:

- **Add a new access policy** - Select Add. Once generated, the **Access Policies** dialog will display the newly added access policy (with default settings).
- **Edit an access policy** - Make any desired edits, and select Save.
- **Remove an access policy** - Select Remove next to the access policy you wish to remove.

#### NOTE

Modifying immutability policies is not supported from Storage Explorer.

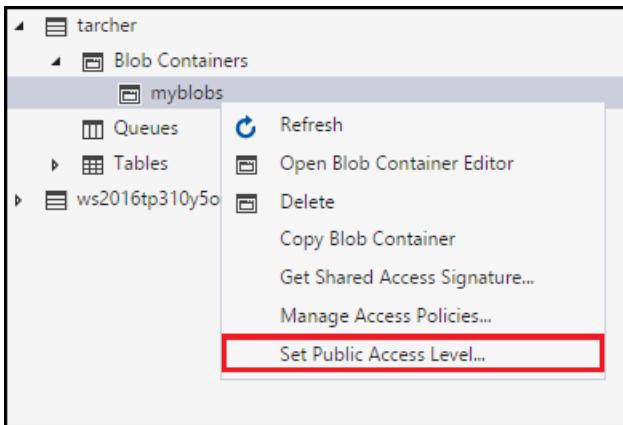
## Set the Public Access Level for a blob container

By default, every blob container is set to "No public access".

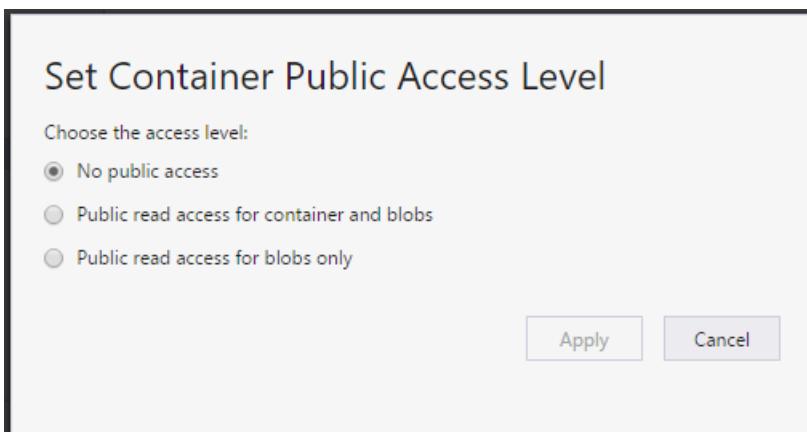
The following steps illustrate how to specify a public access level for a blob container.

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the blob container whose access policies you wish to manage.
3. Expand the storage account's **Blob Containers**.

4. Select the desired blob container, and - from the context menu - select Set Public Access Level.



5. In the Set Container Public Access Level dialog, specify the desired access level.



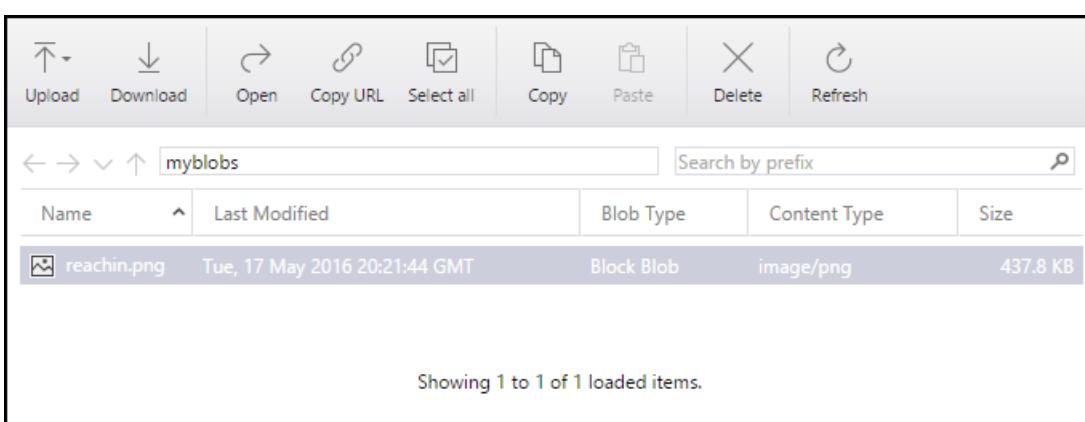
6. Select Apply.

## Managing blobs in a blob container

Once you've created a blob container, you can upload a blob to that blob container, download a blob to your local computer, open a blob on your local computer, and much more.

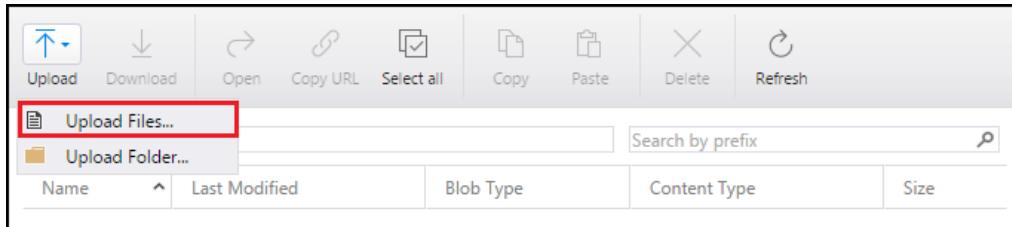
The following steps illustrate how to manage the blobs (and folders) within a blob container.

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the blob container you wish to manage.
3. Expand the storage account's **Blob Containers**.
4. Double-click the blob container you wish to view.
5. The main pane will display the blob container's contents.

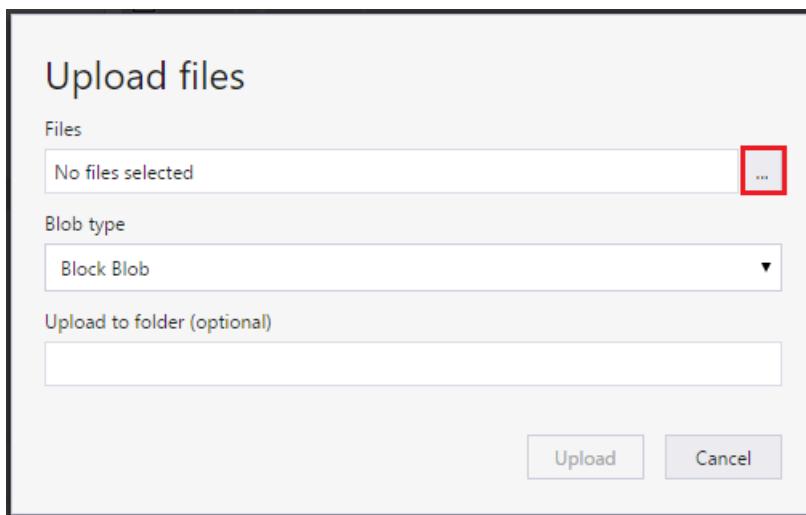


6. The main pane will display the blob container's contents.
7. Follow these steps depending on the task you wish to perform:
  - **Upload files to a blob container**

- a. On the main pane's toolbar, select **Upload**, and then **Upload Files** from the drop-down menu.



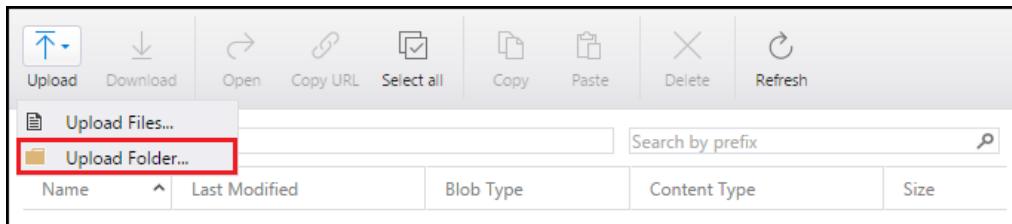
- b. In the **Upload files** dialog, select the ellipsis (...) button on the right side of the **Files** text box to select the file(s) you wish to upload.



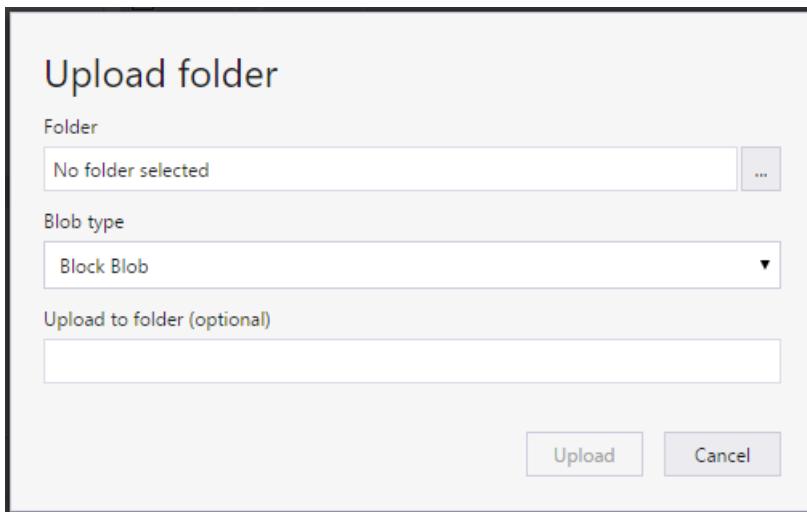
- c. Specify the type of **Blob type**. See [Create a container](#) for more information.
  - d. Optionally, specify a target folder into which the selected file(s) will be uploaded. If the target folder doesn't exist, it will be created.
  - e. Select **Upload**.

- **Upload a folder to a blob container**

- a. On the main pane's toolbar, select **Upload**, and then **Upload Folder** from the drop-down menu.



- b. In the **Upload folder** dialog, select the ellipsis (...) button on the right side of the **Folder** text box to select the folder whose contents you wish to upload.



- c. Specify the type of **Blob type**. See [Create a container](#) for more information.
  - d. Optionally, specify a target folder into which the selected folder's contents will be uploaded.  
If the target folder doesn't exist, it will be created.
  - e. Select **Upload**.
- **Download a blob to your local computer**
    - a. Select the blob you wish to download.
    - b. On the main pane's toolbar, select **Download**.
    - c. In the **Specify where to save the downloaded blob** dialog, specify the location where you want the blob downloaded, and the name you wish to give it.
    - d. Select **Save**.
  - **Open a blob on your local computer**
    - a. Select the blob you wish to open.
    - b. On the main pane's toolbar, select **Open**.
    - c. The blob will be downloaded and opened using the application associated with the blob's underlying file type.
  - **Copy a blob to the clipboard**
    - a. Select the blob you wish to copy.
    - b. On the main pane's toolbar, select **Copy**.
    - c. In the left pane, navigate to another blob container, and double-click it to view it in the main pane.
    - d. On the main pane's toolbar, select **Paste** to create a copy of the blob.
  - **Delete a blob**
    - a. Select the blob you wish to delete.
    - b. On the main pane's toolbar, select **Delete**.
    - c. Select **Yes** to the confirmation dialog.

## Next steps

- View the [latest Storage Explorer release notes and videos](#).
- Learn how to [create applications using Azure blobs, tables, queues, and files](#).

# Using Storage Explorer with Azure Files

8/22/2022 • 7 minutes to read • [Edit Online](#)

Azure Files is a service that offers file shares in the cloud using the standard Server Message Block (SMB) Protocol. Both SMB 2.1 and SMB 3.0 are supported. With Azure Files, you can migrate legacy applications that rely on file shares to Azure quickly and without costly rewrites. You can use File storage to expose data publicly to the world, or to store application data privately. In this article, you'll learn how to use Storage Explorer to work with file shares and files.

## Prerequisites

To complete the steps in this article, you'll need the following:

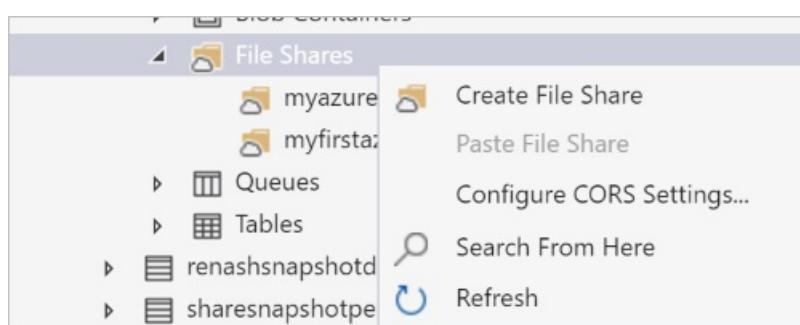
- [Download and install Storage Explorer](#)
- [Connect to an Azure storage account or service](#)

## Create a file share

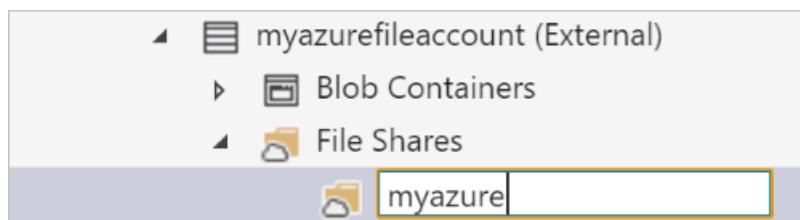
All files must reside in a file share, which is simply a logical grouping of files. An account can contain an unlimited number of file shares, and each share can store an unlimited number of files.

The following steps illustrate how to create a file share within Storage Explorer.

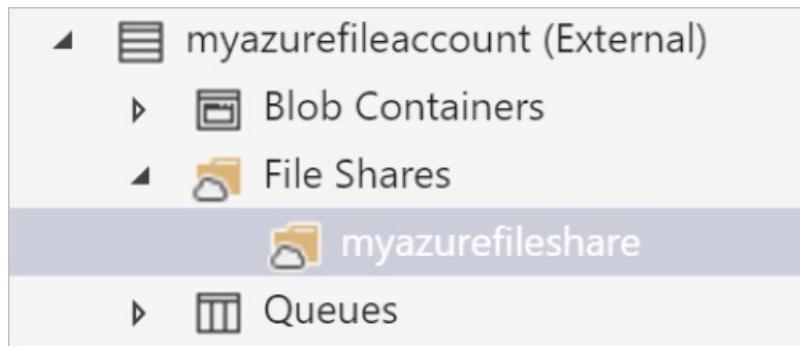
1. Open Storage Explorer.
2. In the left pane, expand the storage account within which you wish to create the file share
3. Right-click **File Shares**, and - from the context menu - select **Create File Share**.



4. A text box will appear below the **File Shares** folder. Enter the name for your file share. See the [Share naming rules](#) section for a list of rules and restrictions on naming file shares.



5. Press **Enter** when done to create the file share, or **Esc** to cancel. Once the file share has been successfully created, it will be displayed under the **File Shares** folder for the selected storage account.

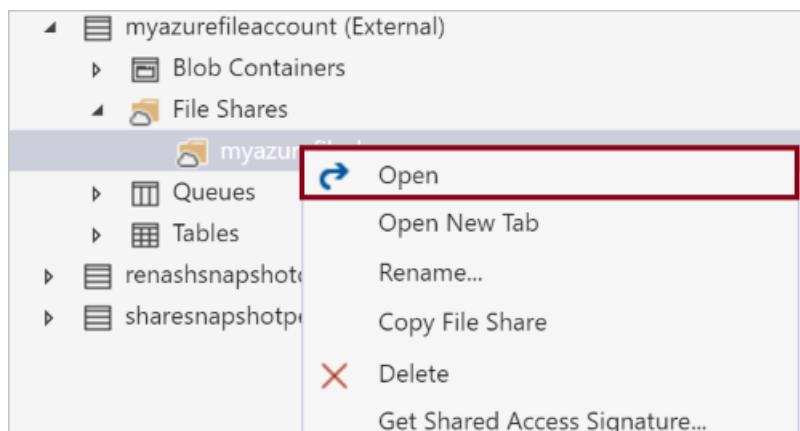


## View a file share's contents

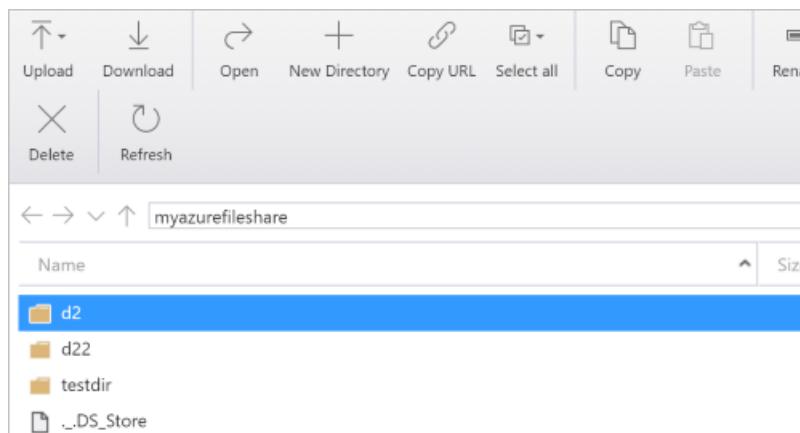
File shares contain files and folders (that can also contain files).

The following steps illustrate how to view the contents of a file share within Storage Explorer:-

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the file share you wish to view.
3. Expand the storage account's **File Shares**.
4. Right-click the file share you wish to view, and - from the context menu - select **Open**. You can also double-click the file share you wish to view.



5. The main pane will display the file share's contents.

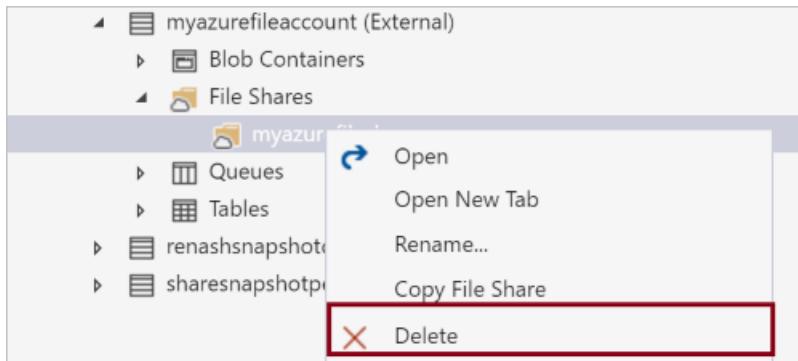


## Delete a file share

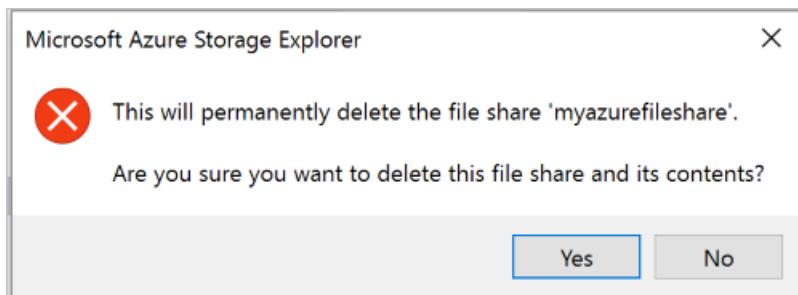
File shares can be easily created and deleted as needed. (To see how to delete individual files, refer to the section, [Managing files in a file share](#).)

The following steps illustrate how to delete a file share within Storage Explorer:

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the file share you wish to view.
3. Expand the storage account's **File Shares**.
4. Right-click the file share you wish to delete, and - from the context menu - select **Delete**. You can also press **Delete** to delete the currently selected file share.



5. Select **Yes** to the confirmation dialog.

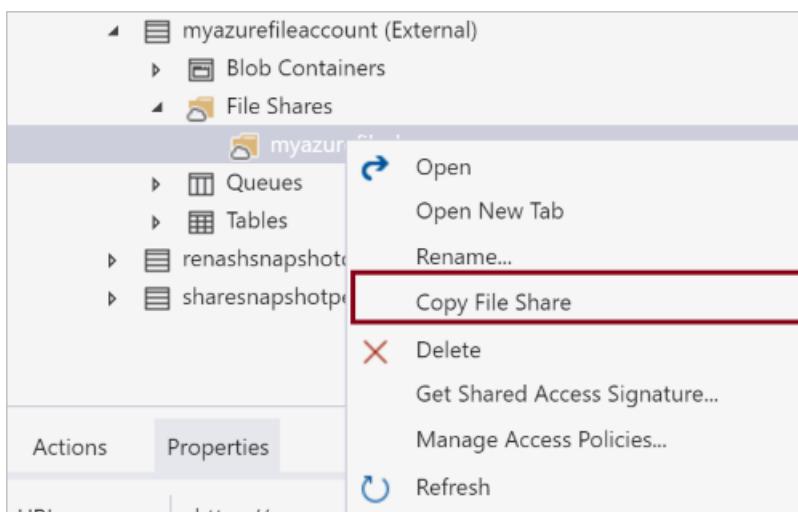


## Copy a file share

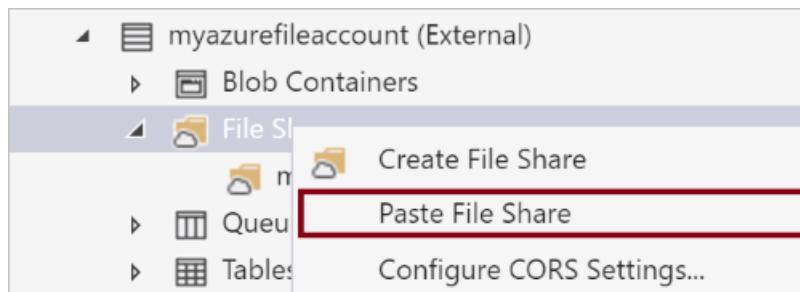
Storage Explorer enables you to copy a file share to the clipboard, and then paste that file share into another storage account. (To see how to copy individual files, refer to the section, [Managing files in a file share](#).)

The following steps illustrate how to copy a file share from one storage account to another.

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the file share you wish to copy.
3. Expand the storage account's **File Shares**.
4. Right-click the file share you wish to copy, and - from the context menu - select **Copy File Share**.



5. Right-click the desired "target" storage account into which you want to paste the file share, and - from the context menu - select **Paste File Share**.

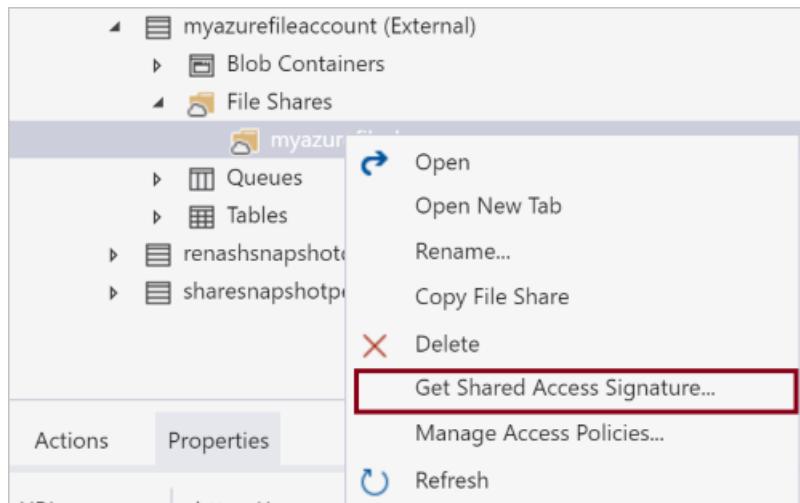


## Get the SAS for a file share

A [shared access signature \(SAS\)](#) provides delegated access to resources in your storage account. This means that you can grant a client limited permissions to objects in your storage account for a specified period of time and with a specified set of permissions, without having to share your account access keys.

The following steps illustrate how to create a SAS for a file share:+

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the file share for which you wish to get a SAS.
3. Expand the storage account's **File Shares**.
4. Right-click the desired file share, and - from the context menu - select **Get Shared Access Signature**.



5. In the **Shared Access Signature** dialog, specify the policy, start and expiration dates, time zone, and access levels you want for the resource.

## Shared Access Signature

Access policy:

Start time:

Expiry time:

Time zone:  
 Local  
 UTC

Permissions:

Read  
 Write  
 Delete  
 List

6. When you're finished specifying the SAS options, select **Create**.
7. A second **Shared Access Signature** dialog will then display that lists the file share along with the URL and QueryStrings you can use to access the storage resource. Select **Copy** next to the URL you wish to copy to the clipboard.

## Shared Access Signature

Share:

URL:

Query string:

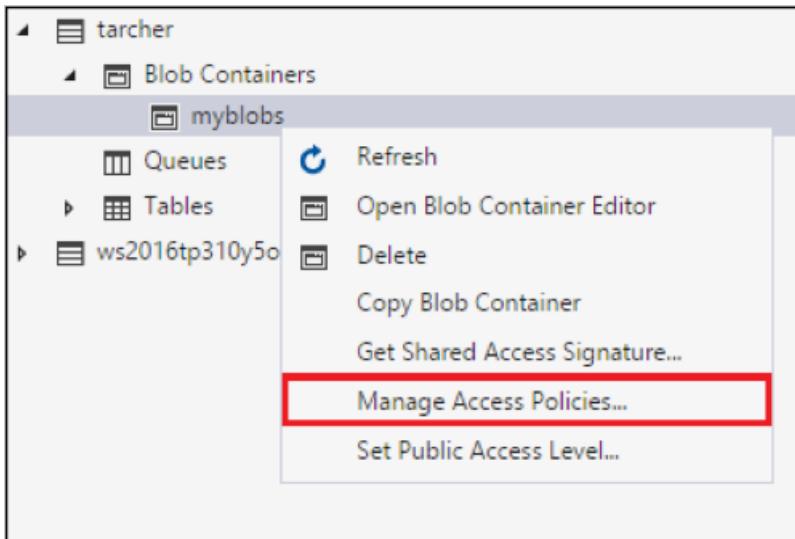
8. When done, select **Close**.

## Manage Access Policies for a file share

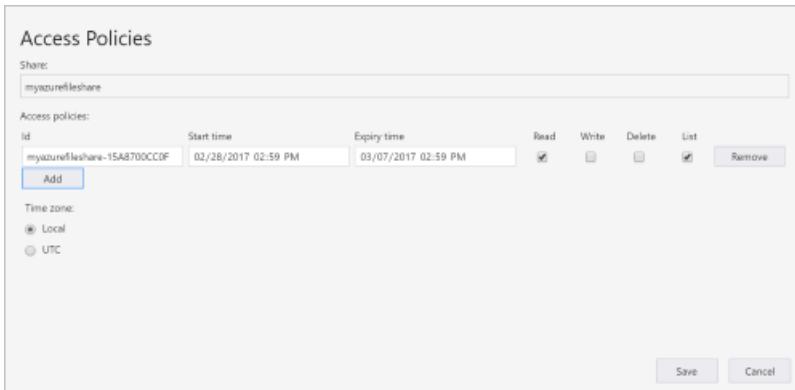
The following steps illustrate how to manage (add and remove) access policies for a file share: . The Access Policies is used for creating SAS URLs through which people can use to access the Azure Files resource during a defined period of time.

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the file share whose access policies you wish to manage.

3. Expand the storage account's **File Shares**.
4. Select the desired file share, and - from the context menu - select **Manage Access Policies**.



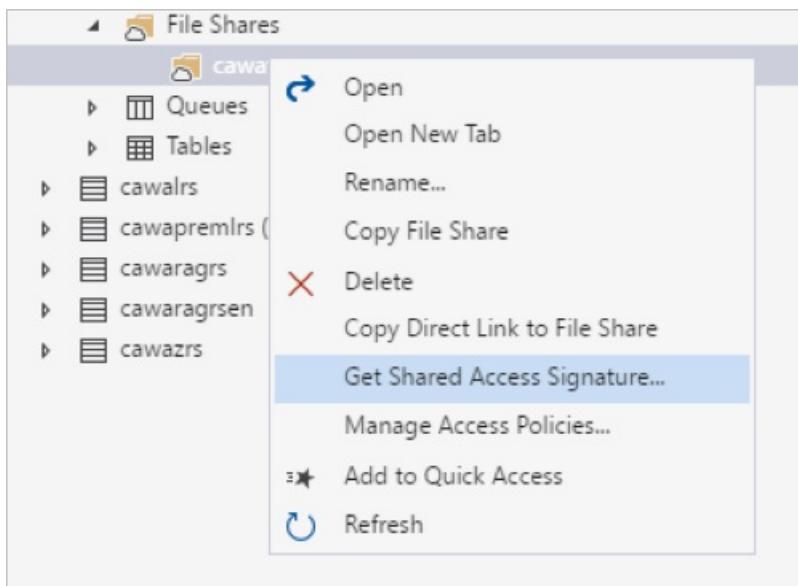
5. The **Access Policies** dialog will list any access policies already created for the selected file share.



6. Follow these steps depending on the access policy management task:

  - **Add a new access policy** - Select **Add**. Once generated, the **Access Policies** dialog will display the newly added access policy (with default settings).
  - **Edit an access policy** - Make any desired edits, and select **Save**.
  - **Remove an access policy** - Select **Remove** next to the access policy you wish to remove.

7. Create a new SAS URL using the Access Policy you created earlier:



### Shared Access Signature

Access policy: myazurefileshare-15A8700CC0F

Start time: 03/06/2017 02:52 PM

Expiry time: 03/13/2017 02:52 PM

Time zone:  
 Local  
 UTC

Permissions:

Read  
 Write  
 Delete  
 List

**Create** **Cancel**

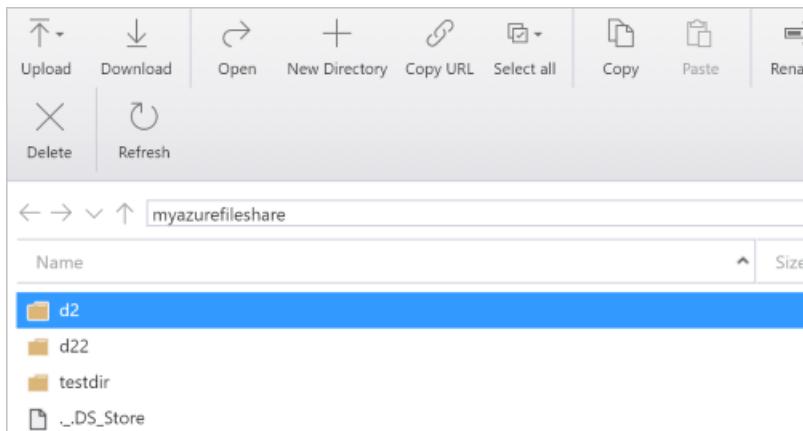
This is a configuration dialog for creating a Shared Access Signature. It includes fields for the access policy name, start and expiry times, time zone (set to Local), and a list of permissions (Read, Write, Delete, List) with checkboxes. At the bottom are 'Create' and 'Cancel' buttons.

## Managing files in a file share

Once you've created a file share, you can upload a file to that file share, download a file to your local computer, open a file on your local computer, and much more.

The following steps illustrate how to manage the files (and folders) within a file share.

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the file share you wish to manage.
3. Expand the storage account's **File Shares**.
4. Double-click the file share you wish to view.
5. The main pane will display the file share's contents.

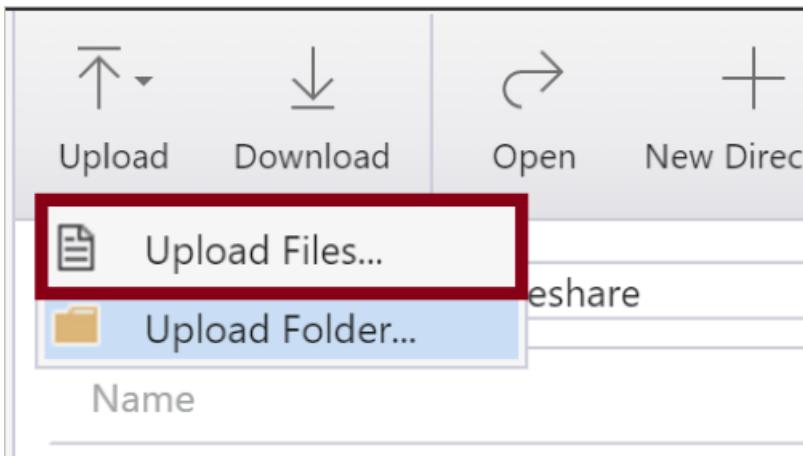


6. The main pane will display the file share's contents.

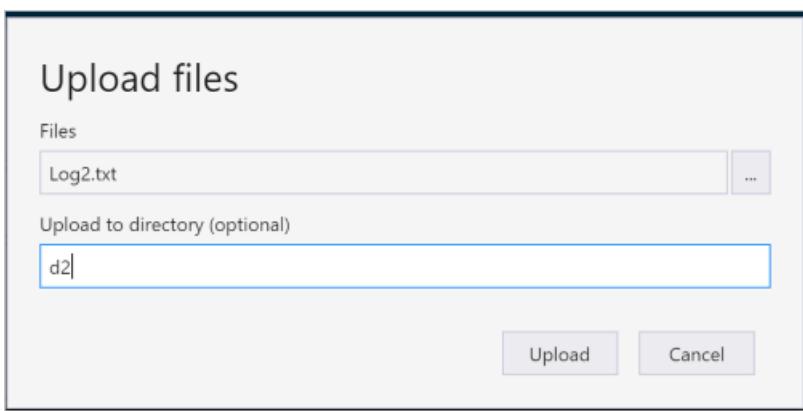
7. Follow these steps depending on the task you wish to perform:

- **Upload files to a file share**

a. On the main pane's toolbar, select **Upload**, and then **Upload Files** from the drop-down menu.



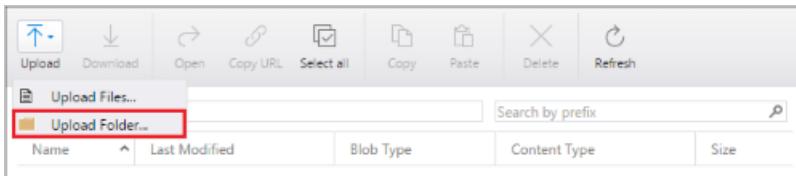
b. In the **Upload files** dialog, select the ellipsis (...) button on the right side of the **Files** text box to select the file(s) you wish to upload.



c. Select **Upload**.

- **Upload a folder to a file share**

a. On the main pane's toolbar, select **Upload**, and then **Upload Folder** from the drop-down menu.



- b. In the **Upload folder** dialog, select the ellipsis (...) button on the right side of the **Folder** text box to select the folder whose contents you wish to upload.
  - c. Optionally, specify a target folder into which the selected folder's contents will be uploaded. If the target folder doesn't exist, it will be created.
  - d. Select **Upload**.
- **Download a file to your local computer**
    - a. Select the file you wish to download.
    - b. On the main pane's toolbar, select **Download**.
    - c. In the **Specify where to save the downloaded file** dialog, specify the location where you want the file downloaded, and the name you wish to give it.
    - d. Select **Save**.
  - **Open a file on your local computer**
    - a. Select the file you wish to open.
    - b. On the main pane's toolbar, select **Open**.
    - c. The file will be downloaded and opened using the application associated with the file's underlying file type.
  - **Copy a file to the clipboard**
    - a. Select the file you wish to copy.
    - b. On the main pane's toolbar, select **Copy**.
    - c. In the left pane, navigate to another file share, and double-click it to view it in the main pane.
    - d. On the main pane's toolbar, select **Paste** to create a copy of the file.
  - **Delete a file**
    - a. Select the file you wish to delete.
    - b. On the main pane's toolbar, select **Delete**.
    - c. Select **Yes** to the confirmation dialog.

## Next steps

- View the [latest Storage Explorer release notes and videos](#).
- Learn how to [create applications using Azure blobs, tables, queues, and files](#).

# Azure Storage Explorer support lifecycle and policy

8/22/2022 • 3 minutes to read • [Edit Online](#)

This document covers the support lifecycle and policy for Azure Storage Explorer.

## Update schedule and process

Azure Storage Explorer is released four to six times a year. Microsoft may also bypass this schedule to release fixes for critical issues.

Storage Explorer checks for updates every hour, and downloads them when they're available. User acceptance is required, however, to start the install process. If users choose to update at a different time, they can manually check for update. On Windows and Linux, users can check for updates by selecting **Check for Updates** from the **Help** menu. On macOS, **Check for Updates** can be found under the **app** menu. Users may also directly go to the [Storage Explorer](#) download page to download and install the latest release.

We strongly recommend to always use the latest versions of Storage Explorer. If an issue is preventing you from updating to the latest version of Storage Explorer, open an issue on our [GitHub](#).

## Support lifecycle

Storage Explorer is governed by the [Modern Lifecycle Policy](#). It's expected that users keep their installation of Storage Explorer up to date. Staying up to date ensures that users have the latest capabilities, performance enhancements, security, and service reliability.

Starting with version 1.14.1, any Storage Explorer release that is greater than 12 months old will be considered out of support. All releases before 1.14.1 will be considered out of support starting on July 14, 2021. Versions that are out of support are no longer guaranteed to work fully as designed and expected. For a list of all releases, their release date, and their end of support date, see [Releases](#).

Starting with version 1.13.0, an in-app alert may be displayed once a version is approximately one month away from being out of support. The alert encourages users to update to the latest version of Storage Explorer. Once a version is out of support, the in-app alert may be displayed on each start-up.

## Releases

This table describes the release date and the end of support date for each release of Azure Storage Explorer.

STORAGE EXPLORER VERSION	RELEASE DATE	END OF SUPPORT DATE
v1.25.0	August 3, 2022	August 3, 2023
v1.24.3	June 21, 2022	June 21, 2023
v1.24.2	May 27, 2022	May 27, 2023
v1.24.1	May 12, 2022	May 12, 2023
v1.24.0	May 3, 2022	May 3, 2023
v1.23.1	April 12, 2022	April 12, 2023

STORAGE EXPLORER VERSION	RELEASE DATE	END OF SUPPORT DATE
v1.23.0	March 2, 2022	March 2, 2023
v1.22.1	January 25, 2022	January 25, 2023
v1.22.0	December 14, 2021	December 14, 2022
v1.21.3	October 25, 2021	October 25, 2022
v1.21.2	September 28, 2021	September 28, 2022
v1.21.1	September 22, 2021	September 22, 2022
v1.21.0	September 8, 2021	September 8, 2022
v1.20.1	July 23, 2021	July 23, 2022
v1.20.0	June 25, 2021	June 25, 2022
v1.19.1	April 29, 2021	April 29, 2022
v1.19.0	April 15, 2021	April 15, 2022
v1.18.1	March 4, 2021	March 4, 2022
v1.18.0	March 1, 2021	March 1, 2022
v1.17.0	December 15, 2020	December 15, 2021
v1.16.0	November 10, 2020	November 10, 2021
v1.15.1	September 2, 2020	September 2, 2021
v1.15.0	August 27, 2020	August 27, 2021
v1.14.2	July 16, 2020	July 16, 2021
v1.14.1	July 14, 2020	July 14, 2021
v1.14.0	June 24, 2020	July 14, 2021
v1.13.1	May 18, 2020	July 14, 2021
v1.13.0	May 1, 2020	July 14, 2021
v1.12.0	January 16, 2020	July 14, 2021
v1.11.2	December 17, 2019	July 14, 2021
v1.11.1	November 20, 2019	July 14, 2021

STORAGE EXPLORER VERSION	RELEASE DATE	END OF SUPPORT DATE
v1.11.0	November 4, 2019	July 14, 2021
v1.10.1	September 19, 2019	July 14, 2021
v1.10.0	September 12, 2019	July 14, 2021
v1.9.0	July 1, 2019	July 14, 2021
v1.8.1	May 10, 2019	July 14, 2021
v1.8.0	May 2, 2019	July 14, 2021
v1.7.0	March 5, 2019	July 14, 2021
v1.6.2	January 8, 2019	July 14, 2021
v1.6.1	December 18, 2018	July 14, 2021
v1.6.0	December 4, 2018	July 14, 2021
v1.5.0	October 29, 2018	July 14, 2021
v1.4.4	October 15, 2018	July 14, 2021
v1.4.2	September 24, 2018	July 14, 2021
v1.4.1	August 28, 2018	July 14, 2021
v1.3.1	July 11, 2018	July 14, 2021
v1.2.0	June 12, 2018	July 14, 2021
v1.1.0	May 9, 2018	July 14, 2021
v1.0.0 (and older)	April 16, 2018	July 14, 2021

# Storage Explorer Accessibility

8/22/2022 • 2 minutes to read • [Edit Online](#)

## Screen Readers

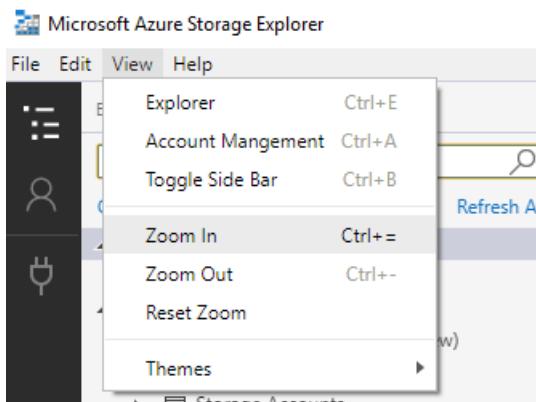
Storage Explorer supports the use of a screen reader on Windows and Mac. The following screen readers are recommended for each platform:

PLATFORM	SCREEN READER
Windows	NVDA
Mac	Voice Over
Linux	(screen readers are not supported on Linux)

If you run into an accessibility issue when using Storage Explorer, please [open an issue on GitHub](#).

## Zoom

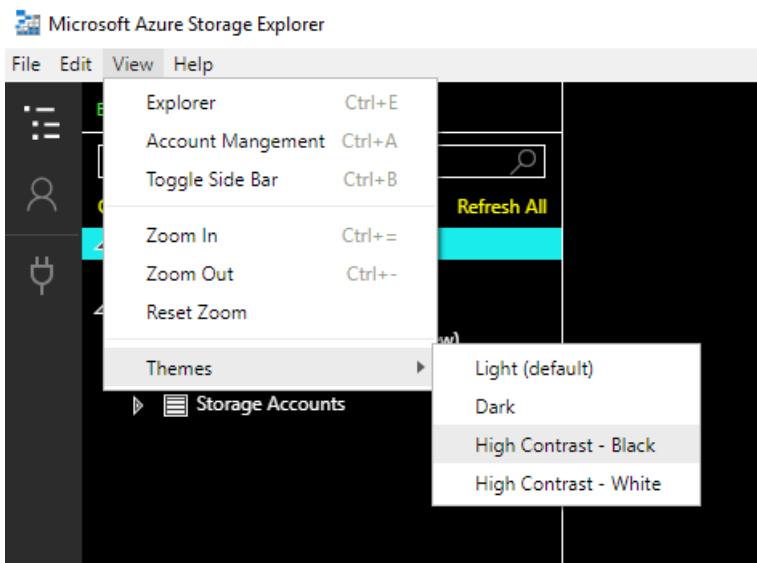
You can make the text in Storage Explorer larger via zooming in. To zoom in, click on **Zoom In** in the Help menu. You can also use the Help menu to zoom out and reset the zoom level back to the default level.



The zoom setting increases the size of most UI elements. It is recommended to also enable large text and zoom settings for your OS to ensure that all UI elements are properly scaled.

## High Contrast Themes

Storage Explorer has two high contrast themes, **High Contrast Light** and **High Contrast Dark**. You can change your theme by selecting in from the Help > Themes menu.



The theme setting changes the color of most UI elements. It is recommended to also enable your OS' matching high contrast theme to ensure that all UI elements are properly colored.

## Shortcut Keys

### Window Commands

COMMAND	KEYBOARD SHORTCUT
New Window	Control+Shift+N
Close Editor	Control+F4
Quit	Control+Shift+W

### Navigation Commands

COMMAND	KEYBOARD SHORTCUT
Focus Next Panel	F6
Focus Previous Panel	Shift+F6
Explorer	Control+Shift+E
Account Management	Control+Shift+A
Toggle Side Bar	Control+B
Activity Log	Control+Shift+L
Actions and Properties	Control+Shift+P
Current Editor	Control+Home
Next Editor	Control+Page Down
Previous Editor	Control+Page Up

## Zoom Commands

COMMAND	KEYBOARD SHORTCUT
Zoom In	Control+=
Zoom Out	Control+-

## Blob and File Share Editor Commands

COMMAND	KEYBOARD SHORTCUT
Back	Alt+Left Arrow
Forward	Alt+Right Arrow
Up	Alt+Up Arrow

## Editor Commands

COMMAND	KEYBOARD SHORTCUT
Copy	Control+C
Cut	Control+X
Paste	Control+V
Refresh	Control+R

## Other Commands

COMMAND	KEYBOARD SHORTCUT
Toggle Developer Tools	F12
Reload	Alt+Control+R

# Compare access to Azure Files, Blob Storage, and Azure NetApp Files with NFS

8/22/2022 • 2 minutes to read • [Edit Online](#)

This article provides a comparison between each of these offerings if you access them through the [Network File System \(NFS\)](#) protocol. This comparison doesn't apply if you access them through any other method.

For more general comparisons, see the [this article](#) to compare Azure Blob Storage and Azure Files, or [this article](#) to compare Azure Files and Azure NetApp Files.

## Comparison

Category	Azure Blob Storage	Azure Files	Azure NetApp Files
Use cases	<p>Blob Storage is best suited for large scale read-heavy sequential access workloads where data is ingested once and minimally modified further.</p> <p>Blob Storage offers the lowest total cost of ownership, if there is little or no maintenance.</p> <p>Some example scenarios are: Large scale analytical data, throughput sensitive high-performance computing, backup and archive, autonomous driving, media rendering, or genomic sequencing.</p>	<p>Azure Files is a highly available service best suited for random access workloads.</p> <p>For NFS shares, Azure Files provides full POSIX file system support and can easily be used from container platforms like Azure Container Instance (ACI) and Azure Kubernetes Service (AKS) with the built-in CSI driver, in addition to VM-based platforms.</p> <p>Some example scenarios are: Shared files, databases, home directories, traditional applications, ERP, CMS, NAS migrations that don't require advanced management, and custom applications requiring scale-out file storage.</p>	<p>Fully managed file service in the cloud, powered by NetApp, with advanced management capabilities.</p> <p>NetApp Files is suited for workloads that require random access and provides broad protocol support and data protection capabilities.</p> <p>Some example scenarios are: On-premises enterprise NAS migration that requires rich management capabilities, latency sensitive workloads like SAP HANA, latency-sensitive or IOPS intensive high performance compute, or workloads that require simultaneous multi-protocol access.</p>
Available protocols	<p>NFS 3.0</p> <p>REST</p> <p>Data Lake Storage Gen2</p>	<p>SMB</p> <p>NFS 4.1</p> <p>(No interoperability between either protocol)</p>	<p>NFS 3.0 and 4.1</p> <p>SMB</p>

Category	Azure Blob Storage	Azure Files	Azure NetApp Files
Key features	<p>Integrated with HPC cache for low latency workloads.</p> <p>Integrated management, including lifecycle, immutable blobs, data failover, and metadata index.</p>	<p>Zonally redundant for high availability.</p> <p>Consistent single-digit millisecond latency.</p> <p>Predictable performance and cost that scales with capacity.</p>	<p>Extremely low latency (as low as sub-ms).</p> <p>Rich NetApp ONTAP management capability such as SnapMirror in cloud.</p> <p>Consistent hybrid cloud experience.</p>
Performance (Per volume)	Up to 20,000 IOPS, up to 100 Gib/s throughput.	Up to 100,000 IOPS, up to 80 Gib/s throughput.	Up to 460,000 IOPS, up to 36 Gib/s throughput.
Scale	<p>Up to 2 PiB for a single volume.</p> <p>Up to ~4.75 TiB max for a single file.</p> <p>No minimum capacity requirements.</p>	<p>Up to 100 TiB for a single file share.</p> <p>Up to 4 TiB for a single file.</p> <p>100 GiB min capacity.</p>	<p>Up to 100 TiB for a single volume.</p> <p>Up to 16 TiB for a single file.</p> <p>Consistent hybrid cloud experience.</p>
Pricing	<a href="#">Azure Blob Storage pricing</a>	<a href="#">Azure Files pricing</a>	<a href="#">Azure NetApp Files pricing</a>

## Next steps

- To access Blob storage with NFS, see [Network File System \(NFS\) 3.0 protocol support in Azure Blob Storage](#).
- To access Azure Files with NFS, see [NFS file shares in Azure Files](#).
- To access Azure NetApp Files with NFS, see [Quickstart: Set up Azure NetApp Files and create an NFS volume](#).

# Azure Storage compliance offerings

8/22/2022 • 2 minutes to read • [Edit Online](#)

To help organizations comply with national, regional, and industry-specific requirements governing the collection and use of individuals' data, Microsoft Azure & Azure Storage offer the most comprehensive set of certifications and attestations of any cloud service provider.

You can find below compliance offerings on Azure Storage to ensure your service regulated in using Azure Storage service. They are applicable to the following Azure Storage offerings: Blobs(ADLS Gen2), Files, Queues, Tables, Disks, Cool Storage, and Premium Storage.

## Global

- [CSA-STAR-Attestation](#)
- [CSA-Star-Certification](#)
- [CSA-STAR-Self-Assessment](#)
- [ISO 20000-1:2011](#)
- [ISO 22301](#)
- [ISO 27001](#)
- [ISO 27017](#)
- [ISO 27018](#)
- [ISO 9001](#)
- [WCAG 2.0](#)

## US Government

- [DoD DISA L2, L4, L5](#)
- [DoE 10 CFR Part 810](#)
- [EAR \(US Export Administration Regulations\)](#)
- [FDA CFR Title 21 Part 11](#)
- [FedRAMP](#)
- [FERPA](#)
- [FIPS 140-2](#)
- [NIST 800-171](#)
- [Section 508 VPATs](#)

## Industry

- [23 NYCRR Part 500](#)
- [APRA \(Australia\)](#)
- [CDSA](#)
- [DPP \(UK\)](#)
- [FACT \(UK\)](#)
- [FCA \(UK\)](#)
- [FFIEC](#)
- [FISC \(Japan\)](#)

- [GLBA](#)
- [GxP](#)
- [HIPAA/HITECH](#)
- [HITRUST](#)
- [MARS-E](#)
- [MAS + ABS \(Singapore\)](#)
- [MPAA](#)
- [NEN-7510 \(Netherlands\)](#)
- [PCI DSS](#)
- [Shared Assessments](#)
- [SOX](#)

## Regional

- [BIR 2012 \(Netherlands\)](#)
- [C5 \(Germany\)](#)
- [CCSL/IRAP \(Australia\)](#)
- [CS Gold Mark \(Japan\)](#)
- [DJCP \(China\)](#)
- [ENISA IAF \(EU\)](#)
- [ENS \(Spain\)](#)
- [EU-Model-Clauses](#)
- [EU-U.S. Privacy Shield](#)
- [GB 18030 \(China\)](#)
- [GDPR \(EU\)](#)
- [IT Grundschutz Workbook \(Germany\)](#)
- [LOPD \(Spain\)](#)
- [MTCS \(Singapore\)](#)
- [My Number \(Japan\)](#)
- [NZ CC Framework \(New Zealand\)](#)
- [PASF \(UK\)](#)
- [PDPA \(Argentina\)](#)
- [PIPEDA \(Canada\)](#)
- [TRUCS \(China\)](#)
- [UK-G-Cloud](#)

## Next steps

Microsoft Azure & Azure Storage keep leading in compliance offerings, you can find the latest coverage and details in [Microsoft TrustCenter](#).



# Introduction to Azure Data Lake Storage Gen2

8/22/2022 • 4 minutes to read • [Edit Online](#)

Azure Data Lake Storage Gen2 is a set of capabilities dedicated to big data analytics, built on [Azure Blob Storage](#).

Data Lake Storage Gen2 converges the capabilities of [Azure Data Lake Storage Gen1](#) with Azure Blob Storage. For example, Data Lake Storage Gen2 provides file system semantics, file-level security, and scale. Because these capabilities are built on Blob storage, you'll also get low-cost, tiered storage, with high availability/disaster recovery capabilities.

## Designed for enterprise big data analytics

Data Lake Storage Gen2 makes Azure Storage the foundation for building enterprise data lakes on Azure. Designed from the start to service multiple petabytes of information while sustaining hundreds of gigabits of throughput, Data Lake Storage Gen2 allows you to easily manage massive amounts of data.

A fundamental part of Data Lake Storage Gen2 is the addition of a [hierarchical namespace](#) to Blob storage. The hierarchical namespace organizes objects/files into a hierarchy of directories for efficient data access. A common object store naming convention uses slashes in the name to mimic a hierarchical directory structure. This structure becomes real with Data Lake Storage Gen2. Operations such as renaming or deleting a directory, become single atomic metadata operations on the directory. There's no need to enumerate and process all objects that share the name prefix of the directory.

Data Lake Storage Gen2 builds on Blob storage and enhances performance, management, and security in the following ways:

- **Performance** is optimized because you do not need to copy or transform data as a prerequisite for analysis. Compared to the flat namespace on Blob storage, the hierarchical namespace greatly improves the performance of directory management operations, which improves overall job performance.
- **Management** is easier because you can organize and manipulate files through directories and subdirectories.
- **Security** is enforceable because you can define POSIX permissions on directories or individual files.

Also, Data Lake Storage Gen2 is very cost effective because it is built on top of the low-cost [Azure Blob Storage](#). The additional features further lower the total cost of ownership for running big data analytics on Azure.

## Key features of Data Lake Storage Gen2

- **Hadoop compatible access:** Data Lake Storage Gen2 allows you to manage and access data just as you would with a [Hadoop Distributed File System \(HDFS\)](#). The new [ABFS driver](#) (used to access data) is available within all Apache Hadoop environments. These environments include [Azure HDInsight](#), [Azure Databricks](#), and [Azure Synapse Analytics](#).
- **A superset of POSIX permissions:** The security model for Data Lake Gen2 supports ACL and POSIX permissions along with some extra granularity specific to Data Lake Storage Gen2. Settings may be configured through Storage Explorer or through frameworks like Hive and Spark.
- **Cost-effective:** Data Lake Storage Gen2 offers low-cost storage capacity and transactions. Features such as [Azure Blob Storage lifecycle](#) optimize costs as data transitions through its lifecycle.
- **Optimized driver:** The ABFS driver is [optimized specifically](#) for big data analytics. The corresponding

REST APIs are surfaced through the endpoint `dfs.core.windows.net`.

## Scalability

Azure Storage is scalable by design whether you access via Data Lake Storage Gen2 or Blob storage interfaces. It is able to store and serve *many exabytes of data*. This amount of storage is available with throughput measured in gigabits per second (Gbps) at high levels of input/output operations per second (IOPS). Processing is executed at near-constant per-request latencies that are measured at the service, account, and file levels.

## Cost effectiveness

Because Data Lake Storage Gen2 is built on top of Azure Blob Storage, storage capacity and transaction costs are lower. Unlike other cloud storage services, you don't have to move or transform your data before you can analyze it. For more information about pricing, see [Azure Storage pricing](#).

Additionally, features such as the [hierarchical namespace](#) significantly improve the overall performance of many analytics jobs. This improvement in performance means that you require less compute power to process the same amount of data, resulting in a lower total cost of ownership (TCO) for the end-to-end analytics job.

## One service, multiple concepts

Because Data Lake Storage Gen2 is built on top of Azure Blob Storage, multiple concepts can describe the same, shared things.

The following are the equivalent entities, as described by different concepts. Unless specified otherwise these entities are directly synonymous:

CONCEPT	TOP LEVEL ORGANIZATION	LOWER LEVEL ORGANIZATION	DATA CONTAINER
Blobs - General purpose object storage	Container	Virtual directory (SDK only - does not provide atomic manipulation)	Blob
Azure Data Lake Storage Gen2 - Analytics Storage	Container	Directory	File

## Supported Blob Storage features

Blob Storage features such as [diagnostic logging](#), [access tiers](#), and [Blob Storage lifecycle management policies](#) are available to your account. Most Blob Storage features are fully supported, but some features are supported only at the preview level or not yet supported.

To see how each Blob Storage feature is supported with Data Lake Storage Gen2, see [Blob Storage feature support in Azure Storage accounts](#).

## Supported Azure service integrations

Data Lake Storage gen2 supports several Azure services. You can use them to ingest data, perform analytics, and create visual representations. For a list of supported Azure services, see [Azure services that support Azure Data Lake Storage Gen2](#).

## Supported open source platforms

Several open source platforms support Data Lake Storage Gen2. For a complete list, see [Open source platforms that support Azure Data Lake Storage Gen2](#).

## See also

- Best practices for using Azure Data Lake Storage Gen2
- Known issues with Azure Data Lake Storage Gen2
- Multi-protocol access on Azure Data Lake Storage

# Blob Storage feature support in Azure Storage accounts

8/22/2022 • 4 minutes to read • [Edit Online](#)

Feature support is impacted by the type of account that you create and the settings that enable on that account. You can use the tables in this article to assess feature support based on these factors. The items that appear in these tables will change over time as support continues to expand.

## How to use these tables

Each table uses the following icons to indicate support level:

ICON	DESCRIPTION
✓	Fully supported
□	Supported at the preview level
□	Not <i>yet</i> supported

This table describes the impact of **enabling** the capability and not the specific use of that capability. For example, if you enable the Network File System (NFS) 3.0 protocol but never use the NFS 3.0 protocol to upload a blob, a check mark in the **NFS 3.0 enabled** column indicates that feature support is not negatively impacted by merely enabling NFS 3.0 support.

Even though a feature is not be negatively impacted, it might not be compatible when used with a specific capability. For example, enabling NFS 3.0 has no impact on Azure Active Directory (Azure AD) authorization. However, you can't use Azure AD to authorize an NFS 3.0 request. See any of these articles for information about known limitations:

- [Known issues: Hierarchical namespace capability](#)
- [Known issues: Network File System \(NFS\) 3.0 protocol](#)
- [Known issues: SSH File Transfer Protocol \(SFTP\)](#)

## Standard general-purpose v2 accounts

The following table describes whether a feature is supported in a standard general-purpose v2 account when you enable a hierarchical namespace (HNS), NFS 3.0 protocol, or SFTP.

### IMPORTANT

This table describes the impact of **enabling** HNS, NFS, or SFTP and not the specific use of those capabilities.

STORAGE FEATURE	DEFAULT	HNS	NFS	SFTP
Access tier - archive	✓	✓	✓	✓

Storage Feature	Default	HNS	NFS	SFTP
Access tier - cool	✓	✓	✓	✓
Access tier - hot	✓	✓	✓	✓
Anonymous public access	✓	✓	✓	✓
Azure Active Directory security	✓	✓	✓ <sup>1</sup>	✓ <sup>1</sup>
Blob inventory	✓	□	□	□
Blob index tags	✓	□	□	□
Blob snapshots	✓	□	□	□
Blob Storage APIs	✓	✓	✓	✓
Blob Storage Azure CLI commands	✓	✓	✓	✓
Blob Storage events	✓	✓	□	✓
Blob Storage PowerShell commands	✓	✓	✓	✓
Blob versioning	✓	□	□	□
Blobfuse	✓	✓	✓	✓
Change feed	✓	□	□	□
Custom domains	✓	□	□	□
Customer-managed account failover	✓	□	□	□
Customer-managed keys (encryption)	✓	✓	✓	✓
Customer-provided keys (encryption)	✓	□	□	□
Data redundancy options	✓	✓	✓ <sup>2</sup>	✓
Encryption scopes	✓	□	□	□
Immutable storage	✓	□	□	□

Storage feature	Default	HNS	NFS	SFTP
Last access time tracking for lifecycle management	✓	✓	□	✓
Lifecycle management policies (delete blob)	✓	✓	✓	✓
Lifecycle management policies (tiering)	✓	✓	✓	✓
Logging in Azure Monitor	✓	✓	□	□
Metrics in Azure Monitor	✓	✓	✓	✓
Object replication for block blobs	✓	□	□	□
Point-in-time restore for block blobs	✓	□	□	□
Soft delete for blobs	✓	✓	✓	✓
Soft delete for containers	✓	✓	✓	✓
Static websites	✓	✓	□	✓
Storage Analytics logs (classic)	✓	✓	□	✓
Storage Analytics metrics (classic)	✓	✓	✓	✓

<sup>1</sup> Requests that clients make by using NFS 3.0 or SFTP can't be authorized by using Azure Active Directory (AD) security.

<sup>2</sup> Only locally redundant storage (LRS) and zone-redundant storage (ZRS) are supported.

## Premium block blob accounts

The following table describes whether a feature is supported in a premium block blob account when you enable a hierarchical namespace (HNS), NFS 3.0 protocol, or SFTP.

### IMPORTANT

This table describes the impact of enabling HNS, NFS, or SFTP and not the specific use of those capabilities.

Storage feature	Default	HNS	NFS	SFTP
Access tier - archive	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Access tier - cool	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Access tier - hot	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Anonymous public access	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Azure Active Directory security	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <sup>1</sup>	<input checked="" type="checkbox"/> <sup>1</sup>
Blob inventory	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Blob index tags	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Blob snapshots	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Blob Storage APIs	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Blob Storage Azure CLI commands	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Blob Storage events	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Blob Storage PowerShell commands	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Blob versioning	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Blobfuse	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Change feed	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Custom domains	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Customer-managed account failover	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Customer-managed keys (encryption)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Customer-provided keys (encryption)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Data redundancy options	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <sup>2</sup>	<input checked="" type="checkbox"/>
Encryption scopes	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Immutable storage	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Storage feature	Default	HNS	NFS	SFTP
Last access time tracking for lifecycle management	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Lifecycle management policies (delete blob)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Lifecycle management policies (tiering)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Logging in Azure Monitor	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Metrics in Azure Monitor	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Object replication for block blobs	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Point-in-time restore for block blobs	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Soft delete for blobs	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Soft delete for containers	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Static websites	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Storage Analytics logs (classic)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Storage Analytics metrics (classic)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

<sup>1</sup> Requests that clients make by using NFS 3.0 or SFTP can't be authorized by using Azure Active Directory (AD) security.

<sup>2</sup> Only locally redundant storage (LRS) and zone-redundant storage (ZRS) are supported.

## See also

- [Known issues with Azure Data Lake Storage Gen2](#)
- [Known issues with Network File System \(NFS\) 3.0 protocol support in Azure Blob Storage](#)
- [Known issues with SSH File Transfer Protocol \(SFTP\) support in Azure Blob Storage \(preview\)](#)

# Tutorial: Query Azure Data Lake Storage Gen2 using SQL language in Synapse Analytics

8/22/2022 • 4 minutes to read • [Edit Online](#)

This tutorial shows you how to connect your Azure Synapse serverless SQL pool to data stored in an Azure storage account that has Azure Data Lake Storage Gen2 enabled. This connection enables you to natively run SQL queries and analytics using SQL language on your data in Azure storage.

In this tutorial, you will:

- Ingest data into a storage account
- Create a Synapse Analytics workspace (if you don't have one).
- Run analytics on your data in Blob storage

If you don't have an Azure subscription, create a [free account](#) before you begin.

## Prerequisites

- Create an Azure Data Lake Storage Gen2 account.

See [Create a storage account to use with Azure Data Lake Storage Gen2](#).

- Make sure that your user account has the [Storage Blob Data Contributor role](#) assigned to it.
- Install AzCopy v10. See [Transfer data with AzCopy v10](#)

There's a couple of specific things that you'll have to do as you perform the steps in that article.

### IMPORTANT

Make sure to assign the role in the scope of the Data Lake Storage Gen2 storage account. You can assign a role to the parent resource group or subscription, but you'll receive permissions-related errors until those role assignments propagate to the storage account.

## Download the flight data

This tutorial uses flight data from the Bureau of Transportation Statistics to demonstrate how to perform an ETL operation. You must download this data to complete the tutorial.

1. Go to [Research and Innovative Technology Administration, Bureau of Transportation Statistics](#).
2. Select the **Prezipped File** check box to select all data fields.
3. Select the **Download** button and save the results to your computer.
4. Unzip the contents of the zipped file and make a note of the file name and the path of the file. You need this information in a later step.

## Copy source data into the storage account

Use AzCopy to copy data from your `.csv` file into your Data Lake Storage Gen2 account.

1. Open a command prompt window, and enter the following command to log into your storage account.

```
azcopy login
```

Follow the instructions that appear in the command prompt window to authenticate your user account.

2. To copy data from the .csv account, enter the following command.

```
azcopy cp "<csv-folder-path>" https://<storage-account-name>.dfs.core.windows.net/<container-name>/folder1/On_Time.csv
```

- Replace the `<csv-folder-path>` placeholder value with the path to the .csv file.
- Replace the `<storage-account-name>` placeholder value with the name of your storage account.
- Replace the `<container-name>` placeholder with the name of a container in your storage account.

## Create an Azure Synapse workspace

In this section, you create an Azure Workspace.

1. Select the **Deploy to Azure** button. The template will open in the Azure portal.



2. Enter or update the following values:

- **Subscription:** Select the Azure subscription where you have the Azure storage account
- **Resource group:** Select the resource group where you placed your Azure Data Lake storage.
- **Region:** Select the region where you placed your Azure Data Lake storage (for example, **Central US**).
- **Name:** Enter a name for your Synapse workspace.
- **SQL Administrator login:** Enter the administrator username for the SQL Server.
- **SQL Administrator password:** Enter the administrator password for the SQL Server.
- **Tag Values:** Accept the default.
- **Review and Create:** Select.
- **Create:** Select.

When the deployment finishes, you will see Azure Synapse Analytics workspace in the list of the deployed resources. You can follow the link to see more details about the workspace and [find your Synapse SQL endpoint name](#).

## Find your Synapse SQL endpoint name

The server name for the serverless SQL pool in the following example is:

`showdemoweu-on-demand.sql.azuresynapse.net`. To find the fully qualified server name:

1. Select on the workspace you want to connect to.
2. Go to overview.
3. Locate the full server name.

## Connect to Synapse SQL endpoint

Synapse SQL endpoint enables you to connect with any tool that can run T-SQL queries on SQL server or Azure SQL database. The examples are [SQL Server Management Studio](#), [Azure Data Studio](#), or [Power BI](#).

Use a tool that you prefer to use to connect to SQL endpoint, put the serverless SQL serverless endpoint name, and connect with Azure AD authentication to connect.

### IMPORTANT

Do not use SQL authentication with username nad password because this will require additional steps to enable SQL login to access your Azure storage account.

## Explore data

Create a new SQL query using the tool that you used to connect to your Synapse endpoint, put the following query, and set the path in

```
SELECT
 TOP 100 *
FROM
 OPENROWSET(
 BULK 'https://<storage-account-name>.dfs.core.windows.net/<container-name>/folder1/On_Time.csv',
 FORMAT='CSV',
 PARSER_VERSION='2.0'
) AS [result]
```

When you execute the query, you will see the content of the file.

## Clean up resources

When they're no longer needed, delete your Synapse Analytics workspace. The workspace tat do not have some additional dedicated SQL pools or Spark pools is not charged if you are not using it, so you will get **no billing even if you keep it**. Do not delete the resource group if you have selected the resource group where you have placed your Azure storage account.

## Next steps

[Azure Data Lake Storage Gen2, Azure Databricks & Spark](#)

# Tutorial: Azure Data Lake Storage Gen2, Azure Databricks & Spark

8/22/2022 • 7 minutes to read • [Edit Online](#)

This tutorial shows you how to connect your Azure Databricks cluster to data stored in an Azure storage account that has Azure Data Lake Storage Gen2 enabled. This connection enables you to natively run queries and analytics from your cluster on your data.

In this tutorial, you will:

- Create a Databricks cluster
- Ingest unstructured data into a storage account
- Run analytics on your data in Blob storage

If you don't have an Azure subscription, create a [free account](#) before you begin.

## Prerequisites

- Create an Azure Data Lake Storage Gen2 account.  
See [Create a storage account to use with Azure Data Lake Storage Gen2](#).
- Make sure that your user account has the [Storage Blob Data Contributor role](#) assigned to it.
- Install AzCopy v10. See [Transfer data with AzCopy v10](#)
- Create a service principal. See [How to: Use the portal to create an Azure AD application and service principal that can access resources](#).

There's a couple of specific things that you'll have to do as you perform the steps in that article.

- ✓ When performing the steps in the [Assign the application to a role](#) section of the article, make sure to assign the [Storage Blob Data Contributor](#) role to the service principal.

### IMPORTANT

Make sure to assign the role in the scope of the Data Lake Storage Gen2 storage account. You can assign a role to the parent resource group or subscription, but you'll receive permissions-related errors until those role assignments propagate to the storage account.

- ✓ When performing the steps in the [Get values for signing in](#) section of the article, paste the tenant ID, app ID, and client secret values into a text file. You'll need those soon.

## Download the flight data

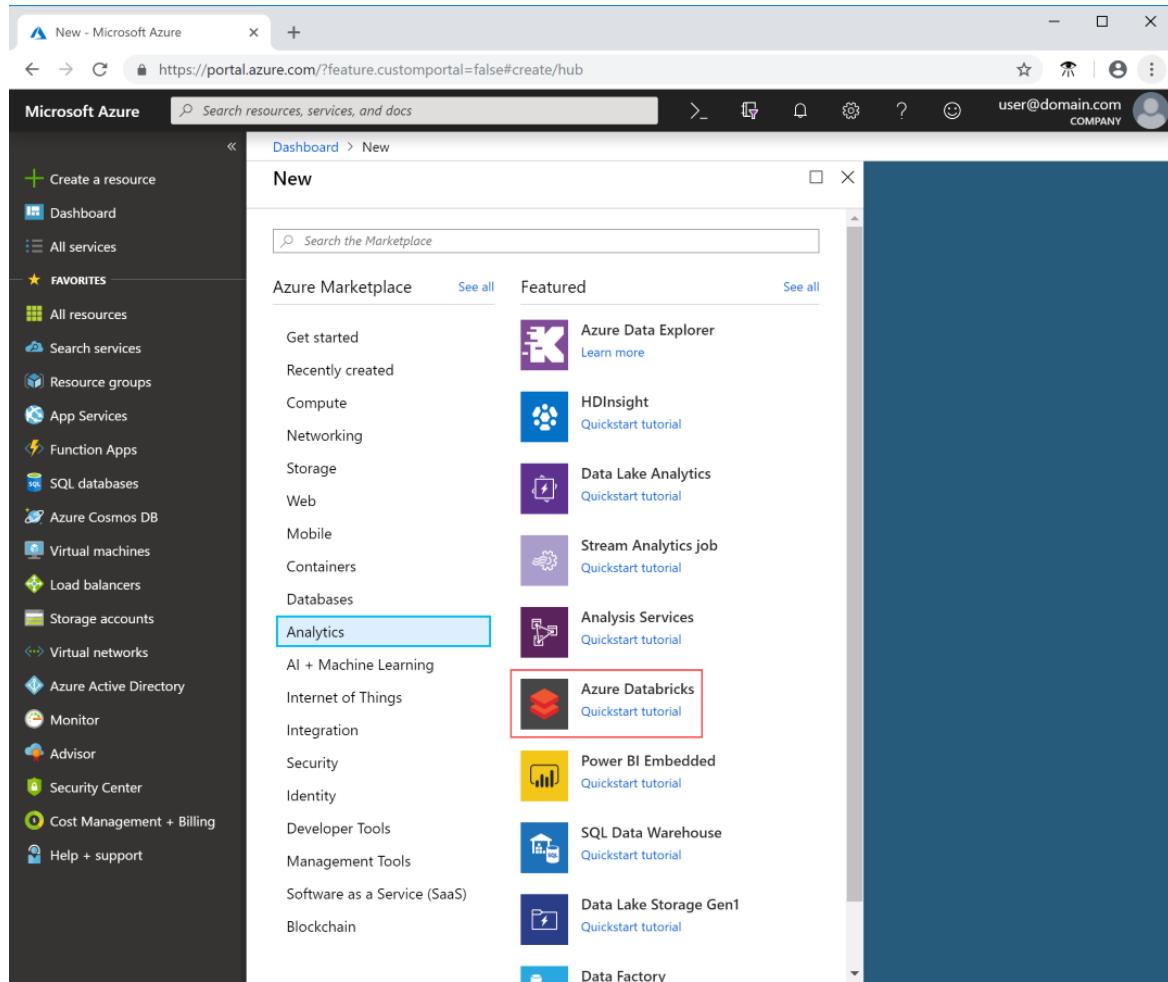
This tutorial uses flight data from the Bureau of Transportation Statistics to demonstrate how to perform an ETL operation. You must download this data to complete the tutorial.

1. Download the [On\\_Time\\_Report\\_Carrier\\_On\\_Time\\_Performance\\_1987\\_present\\_2016\\_1.zip](#) file. This file contains the flight data.
2. Unzip the contents of the zipped file and make a note of the file name and the path of the file. You need this information in a later step.

# Create an Azure Databricks service

In this section, you create an Azure Databricks service by using the Azure portal.

1. In the Azure portal, select **Create a resource > Analytics > Azure Databricks**.



The screenshot shows the Microsoft Azure portal's 'New' blade. On the left, a sidebar lists various service categories like 'Dashboard', 'All services', 'FAVORITES', and 'Analytics'. Under 'Analytics', 'Azure Databricks' is listed and highlighted with a red box. The main content area shows a 'Featured' section with several service cards, including 'Azure Data Explorer', 'HDInsight', 'Data Lake Analytics', 'Stream Analytics job', 'Analysis Services', 'Azure Databricks' (which is also highlighted with a red box), 'Power BI Embedded', 'SQL Data Warehouse', 'Data Lake Storage Gen1', and 'Data Factory'. Each card includes a small icon, the service name, and a 'Quickstart tutorial' link.

2. Under **Azure Databricks Service**, provide the following values to create a Databricks service:

PROPERTY	DESCRIPTION
<b>Workspace name</b>	Provide a name for your Databricks workspace.
<b>Subscription</b>	From the drop-down, select your Azure subscription.
<b>Resource group</b>	Specify whether you want to create a new resource group or use an existing one. A resource group is a container that holds related resources for an Azure solution. For more information, see <a href="#">Azure Resource Group overview</a> .
<b>Location</b>	Select <b>West US 2</b> . For other available regions, see <a href="#">Azure services available by region</a> .
<b>Pricing Tier</b>	Select <b>Standard</b> .

The screenshot shows the 'Azure Databricks Service' creation wizard. The 'Workspace name' field is set to 'mydatabricksws'. The 'Subscription' dropdown is open. Under 'Resource group', 'Create new' is selected and 'mydatabricksresgrp' is chosen. The 'Location' is set to 'West US 2'. The 'Pricing Tier' dropdown shows 'Standard (Apache Spark, Secure with Azure AD)' is selected. Below the pricing tier, there's a note about 'Premium (+ Role-based access controls)'. At the bottom, the 'Pin to dashboard' checkbox is checked, and there are 'Create' and 'Automation options' buttons.

3. The account creation takes a few minutes. To monitor the operation status, view the progress bar at the top.
4. Select **Pin to dashboard** and then select **Create**.

## Create a Spark cluster in Azure Databricks

1. In the Azure portal, go to the Databricks service that you created, and select **Launch Workspace**.
2. You're redirected to the Azure Databricks portal. From the portal, select **Cluster**.

The screenshot shows the Azure Databricks workspace homepage. At the top is the 'Azure Databricks' logo. Below it is a 'Featured Notebooks' section with three items: 'Introduction to Apache Spark on Databricks' (with a Python logo), 'Databricks for Data Scientists' (with a spark icon), and 'Introduction to Structured Streaming' (with a Python logo). Below this is a 'New' section with a 'Cluster' option highlighted with a red box. There are also 'Documentation' and 'Open Recent' sections.

3. In the **New cluster** page, provide the values to create a cluster.

Fill in values for the following fields, and accept the default values for the other fields:

- Enter a name for the cluster.
  - Make sure you select the **Terminate after 120 minutes of inactivity** checkbox. Provide a duration (in minutes) to terminate the cluster, if the cluster is not being used.
4. Select **Create cluster**. After the cluster is running, you can attach notebooks to the cluster and run Spark jobs.

## Ingest data

### Copy source data into the storage account

Use AzCopy to copy data from your .csv file into your Data Lake Storage Gen2 account.

- Open a command prompt window, and enter the following command to log into your storage account.

```
azcopy login
```

Follow the instructions that appear in the command prompt window to authenticate your user account.

- To copy data from the .csv account, enter the following command.

```
azcopy cp "<csv-folder-path>" https://<storage-account-name>.dfs.core.windows.net/<container-name>/folder1/On_Time.csv
```

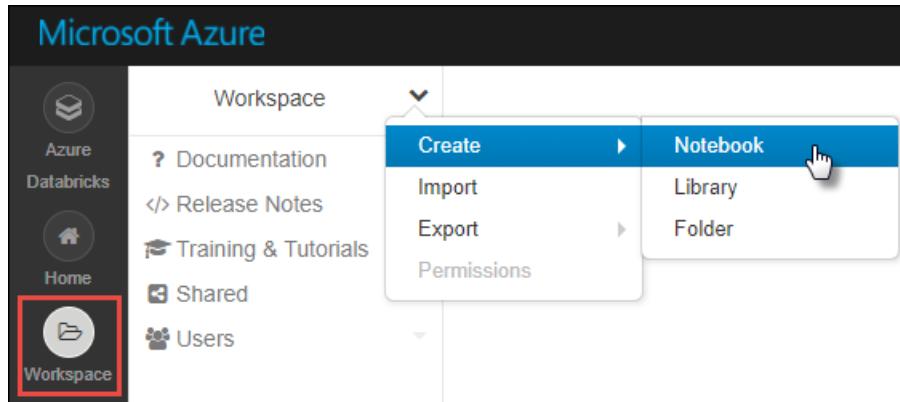
- Replace the <csv-folder-path> placeholder value with the path to the .csv file.

- Replace the <storage-account-name> placeholder value with the name of your storage account.
- Replace the <container-name> placeholder with the name of a container in your storage account.

## Create a container and mount it

In this section, you'll create a container and a folder in your storage account.

1. In the [Azure portal](#), go to the Azure Databricks service that you created, and select **Launch Workspace**.
2. On the left, select **Workspace**. From the **Workspace** drop-down, select **Create > Notebook**.



3. In the **Create Notebook** dialog box, enter a name for the notebook. Select **Python** as the language, and then select the Spark cluster that you created earlier.
4. Select **Create**.
5. Copy and paste the following code block into the first cell, but don't run this code yet.

```
configs = {"fs.azure.account.auth.type": "OAuth",
 "fs.azure.account.oauth.provider.type":
 "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
 "fs.azure.account.oauth2.client.id": "<appId>",
 "fs.azure.account.oauth2.client.secret": "<clientSecret>",
 "fs.azure.account.oauth2.client.endpoint":
 "https://login.microsoftonline.com/<tenant>/oauth2/token",
 "fs.azure.createRemoteFileSystemDuringInitialization": "true"}

dbutils.fs.mount(
 source = "abfss://<container-name>@<storage-account-name>.dfs.core.windows.net/folder1",
 mount_point = "/mnt/flightdata",
 extra_configs = configs)
```

6. In this code block, replace the `<appId>`, `<clientSecret>`, `<tenant>`, and `<storage-account-name>` placeholder values in this code block with the values that you collected while completing the prerequisites of this tutorial. Replace the `<container-name>` placeholder value with the name of the container.
7. Press the **SHIFT + ENTER** keys to run the code in this block.

Keep this notebook open as you will add commands to it later.

### Use Databricks Notebook to convert CSV to Parquet

In the notebook that you previously created, add a new cell, and paste the following code into that cell.

```
Use the previously established DBFS mount point to read the data.
create a data frame to read data.

flightDF = spark.read.format('csv').options(
 header='true', inferSchema='true').load("/mnt/flightdata/*.csv")

read the airline csv file and write the output to parquet format for easy query.
flightDF.write.mode("append").parquet("/mnt/flightdata/parquet/flights")
print("Done")
```

## Explore data

In a new cell, paste the following code to get a list of CSV files uploaded via AzCopy.

```
import os.path
import IPython
from pyspark.sql import SQLContext
display(dbutils.fs.ls("/mnt/flightdata"))
```

To create a new file and list files in the *parquet/flights* folder, run this script:

```
dbutils.fs.put("/mnt/flightdata/1.txt", "Hello, World!", True)
dbutils.fs.ls("/mnt/flightdata/parquet/flights")
```

With these code samples, you have explored the hierarchical nature of HDFS using data stored in a storage account with Data Lake Storage Gen2 enabled.

## Query the data

Next, you can begin to query the data you uploaded into your storage account. Enter each of the following code blocks into **Cmd 1** and press **Cmd + Enter** to run the Python script.

To create data frames for your data sources, run the following script:

- Replace the `<csv-folder-path>` placeholder value with the path to the *.csv* file.

```

Copy this into a Cmd cell in your notebook.
acDF = spark.read.format('csv').options(
 header='true', inferSchema='true').load("/mnt/flightdata/On_Time.csv")
acDF.write.parquet('/mnt/flightdata/parquet/airlinecodes')

read the existing parquet file for the flights database that was created earlier
flightDF = spark.read.format('parquet').options(
 header='true', inferSchema='true').load("/mnt/flightdata/parquet/flights")

print the schema of the dataframes
acDF.printSchema()
flightDF.printSchema()

print the flight database size
print("Number of flights in the database: ", flightDF.count())

show the first 20 rows (20 is the default)
to show the first n rows, run: df.show(n)
acDF.show(100, False)
flightDF.show(20, False)

Display to run visualizations
preferably run this in a separate cmd cell
display(flightDF)

```

Enter this script to run some basic analysis queries against the data.

```

Run each of these queries, preferably in a separate cmd cell for separate analysis
create a temporary sql view for querying flight information
FlightTable = spark.read.parquet('/mnt/flightdata/parquet/flights')
FlightTable.createOrReplaceTempView('FlightTable')

create a temporary sql view for querying airline code information
AirlineCodes = spark.read.parquet('/mnt/flightdata/parquet/airlinecodes')
AirlineCodes.createOrReplaceTempView('AirlineCodes')

using spark sql, query the parquet file to return total flights in January and February 2016
out1 = spark.sql("SELECT * FROM FlightTable WHERE Month=1 and Year= 2016")
NumJan2016Flights = out1.count()
out2 = spark.sql("SELECT * FROM FlightTable WHERE Month=2 and Year= 2016")
NumFeb2016Flights = out2.count()
print("Jan 2016: ", NumJan2016Flights, " Feb 2016: ", NumFeb2016Flights)
Total = NumJan2016Flights+NumFeb2016Flights
print("Total flights combined: ", Total)

List out all the airports in Texas
out = spark.sql(
 "SELECT distinct(OriginCityName) FROM FlightTable where OriginStateName = 'Texas'")
print('Airports in Texas: ', out.show(100))

find all airlines that fly from Texas
out1 = spark.sql(
 "SELECT distinct(Reporting_Airline) FROM FlightTable WHERE OriginStateName='Texas'")
print('Airlines that fly to/from Texas: ', out1.show(100, False))

```

## Clean up resources

When they're no longer needed, delete the resource group and all related resources. To do so, select the resource group for the storage account and select **Delete**.

## Next steps

Extract, transform, and load data using Apache Hive on Azure HDInsight

# Tutorial: Extract, transform, and load data by using Azure HDInsight

8/22/2022 • 7 minutes to read • [Edit Online](#)

In this tutorial, you perform an ETL operation: extract, transform, and load data. You take a raw CSV data file, import it into an Azure HDInsight cluster, transform it with Apache Hive, and load it into Azure SQL Database with Apache Sqoop.

In this tutorial, you learn how to:

- Extract and upload the data to an HDInsight cluster.
- Transform the data by using Apache Hive.
- Load the data to Azure SQL Database by using Sqoop.

If you don't have an Azure subscription, [create a free account](#) before you begin.

## Prerequisites

- An **Azure Data Lake Storage Gen2 storage account** that is configured for HDInsight  
See [Use Azure Data Lake Storage Gen2 with Azure HDInsight clusters](#).
- A **Linux-based Hadoop cluster** on HDInsight  
See [Quickstart: Get started with Apache Hadoop and Apache Hive in Azure HDInsight using the Azure portal](#).
- **Azure SQL Database**: You use Azure SQL Database as a destination data store. If you don't have a database in SQL Database, see [Create a database in Azure SQL Database in the Azure portal](#).
- **Azure CLI**: If you haven't installed the Azure CLI, see [Install the Azure CLI](#).
- **A Secure Shell (SSH) client**: For more information, see [Connect to HDInsight \(Hadoop\) by using SSH](#).

## Download, extract and then upload the data

In this section, you'll download sample flight data. Then, you'll upload that data to your HDInsight cluster and then copy that data to your Data Lake Storage Gen2 account.

1. Download the [On\\_Time\\_Report\\_Carrier\\_On\\_Time\\_Performance\\_1987\\_present\\_2016\\_1.zip](#) file. This file contains the flight data.
2. Open a command prompt and use the following Secure Copy (Scp) command to upload the .zip file to the HDInsight cluster head node:

```
scp <file-name>.zip <ssh-user-name>@<cluster-name>-ssh.azurehdinsight.net:<file-name.zip>
```

- Replace the `<file-name>` placeholder with the name of the .zip file.
- Replace the `<ssh-user-name>` placeholder with the SSH login for the HDInsight cluster.
- Replace the `<cluster-name>` placeholder with the name of the HDInsight cluster.

If you use a password to authenticate your SSH login, you're prompted for the password.

If you use a public key, you might need to use the `-i` parameter and specify the path to the matching private key. For example,

```
scp -i ~/.ssh/id_rsa <file_name>.zip <user-name>@<cluster-name>-ssh.azurehdinsight.net: .
```

3. After the upload has finished, connect to the cluster by using SSH. On the command prompt, enter the following command:

```
ssh <ssh-user-name>@<cluster-name>-ssh.azurehdinsight.net
```

4. Use the following command to unzip the .zip file:

```
unzip <file-name>.zip
```

The command extracts a .csv file.

5. Use the following command to create the Data Lake Storage Gen2 container.

```
hadoop fs -D "fs.azure.createRemoteFileSystemDuringInitialization=true" -ls abfs://<container-name>@<storage-account-name>.dfs.core.windows.net/
```

Replace the `<container-name>` placeholder with the name that you want to give your container.

Replace the `<storage-account-name>` placeholder with the name of your storage account.

6. Use the following command to create a directory.

```
hdfs dfs -mkdir -p abfs://<container-name>@<storage-account-name>.dfs.core.windows.net/tutorials/flightdelays/data
```

7. Use the following command to copy the .csv file to the directory:

```
hdfs dfs -put "<file-name>.csv" abfs://<container-name>@<storage-account-name>.dfs.core.windows.net/tutorials/flightdelays/data/
```

Use quotes around the file name if the file name contains spaces or special characters.

## Transform the data

In this section, you use Beeline to run an Apache Hive job.

As part of the Apache Hive job, you import the data from the .csv file into an Apache Hive table named **delays**.

1. From the SSH prompt that you already have for the HDInsight cluster, use the following command to create and edit a new file named **flightdelays.hql**:

```
nano flightdelays.hql
```

2. Modify the following text by replace the `<container-name>` and `<storage-account-name>` placeholders with your container and storage account name. Then copy and paste the text into the nano console by using pressing the SHIFT key along with the right-mouse click button.

```

DROP TABLE delays_raw;
-- Creates an external table over the csv file
CREATE EXTERNAL TABLE delays_raw (
 YEAR string,
 FL_DATE string,
 UNIQUE_CARRIER string,
 CARRIER string,
 FL_NUM string,
 ORIGIN_AIRPORT_ID string,
 ORIGIN string,
 ORIGIN_CITY_NAME string,
 ORIGIN_CITY_NAME_TEMP string,
 ORIGIN_STATE_ABR string,
 DEST_AIRPORT_ID string,
 DEST string,
 DEST_CITY_NAME string,
 DEST_CITY_NAME_TEMP string,
 DEST_STATE_ABR string,
 DEP_DELAY_NEW float,
 ARR_DELAY_NEW float,
 CARRIER_DELAY float,
 WEATHER_DELAY float,
 NAS_DELAY float,
 SECURITY_DELAY float,
 LATE_AIRCRAFT_DELAY float)
-- The following lines describe the format and location of the file
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
LOCATION 'abfs://<container-name>@<storage-account-
name>.dfs.core.windows.net/tutorials/flightdelays/data';

-- Drop the delays table if it exists
DROP TABLE delays;
-- Create the delays table and populate it with data
-- pulled in from the CSV file (via the external table defined previously)
CREATE TABLE delays
LOCATION 'abfs://<container-name>@<storage-account-
name>.dfs.core.windows.net/tutorials/flightdelays/processed'
AS
SELECT YEAR AS year,
 FL_DATE AS flight_date,
 substring(UNIQUE_CARRIER, 2, length(UNIQUE_CARRIER) -1) AS unique_carrier,
 substring(CARRIER, 2, length(CARRIER) -1) AS carrier,
 substring(FL_NUM, 2, length(FL_NUM) -1) AS flight_num,
 ORIGIN_AIRPORT_ID AS origin_airport_id,
 substring(ORIGIN, 2, length(ORIGIN) -1) AS origin_airport_code,
 substring(ORIGIN_CITY_NAME, 2) AS origin_city_name,
 substring(ORIGIN_STATE_ABR, 2, length(ORIGIN_STATE_ABR) -1) AS origin_state_abr,
 DEST_AIRPORT_ID AS dest_airport_id,
 substring(DEST, 2, length(DEST) -1) AS dest_airport_code,
 substring(DEST_CITY_NAME, 2) AS dest_city_name,
 substring(DEST_STATE_ABR, 2, length(DEST_STATE_ABR) -1) AS dest_state_abr,
 DEP_DELAY_NEW AS dep_delay_new,
 ARR_DELAY_NEW AS arr_delay_new,
 CARRIER_DELAY AS carrier_delay,
 WEATHER_DELAY AS weather_delay,
 NAS_DELAY AS nas_delay,
 SECURITY_DELAY AS security_delay,
 LATE_AIRCRAFT_DELAY AS late_aircraft_delay
FROM delays_raw;

```

3. Save the file by using use CTRL+X and then type  when prompted.

4. To start Hive and run the `flightdelays.hql` file, use the following command:

```
beeline -u 'jdbc:hive2://localhost:10001;transportMode=http' -f flightdelays.hql
```

5. After the `flightdelays.hql` script finishes running, use the following command to open an interactive Beeline session:

```
beeline -u 'jdbc:hive2://localhost:10001;transportMode=http'
```

6. When you receive the `jdbc:hive2://localhost:10001/` prompt, use the following query to retrieve data from the imported flight delay data:

```
INSERT OVERWRITE DIRECTORY '/tutorials/flightdelays/output'
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
SELECT regexp_replace(origin_city_name, ' ', ''),
 avg(weather_delay)
 FROM delays
 WHERE weather_delay IS NOT NULL
 GROUP BY origin_city_name;
```

This query retrieves a list of cities that experienced weather delays, along with the average delay time, and saves it to

`abfs://<container-name>@<storage-account-name>.dfs.core.windows.net/tutorials/flightdelays/output`.

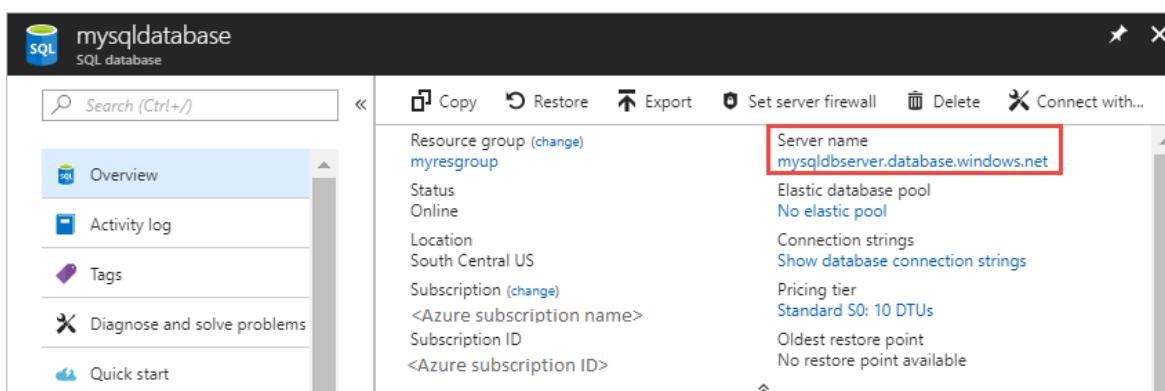
Later, Sqoop reads the data from this location and exports it to Azure SQL Database.

7. To exit Beeline, enter `!quit` at the prompt.

## Create a SQL database table

You need the server name from SQL Database for this operation. Complete these steps to find your server name.

1. Go to the [Azure portal](#).
2. Select **SQL Databases**.
3. Filter on the name of the database that you choose to use. The server name is listed in the **Server name** column.
4. Filter on the name of the database that you want to use. The server name is listed in the **Server name** column.



There are many ways to connect to SQL Database and create a table. The following steps use [FreeTDS](#) from the HDInsight cluster.

5. To install FreeTDS, use the following command from an SSH connection to the cluster:

```
sudo apt-get --assume-yes install freetds-dev freetds-bin
```

6. After the installation completes, use the following command to connect to SQL Database.

```
TDSVER=8.0 tsql -H '<server-name>.database.windows.net' -U '<admin-login>' -p 1433 -D '<database-name>'
```

- Replace the `<server-name>` placeholder with the logical SQL server name.
- Replace the `<admin-login>` placeholder with the admin login for SQL Database.
- Replace the `<database-name>` placeholder with the database name

When you're prompted, enter the password for the SQL Database admin login.

You receive output similar to the following text:

```
locale is "en_US.UTF-8"
locale charset is "UTF-8"
using default charset "UTF-8"
Default database being set to sqooptest
1>
```

7. At the `1>` prompt, enter the following statements:

```
CREATE TABLE [dbo].[delays](
[OriginCityName] [nvarchar](50) NOT NULL,
[WeatherDelay] float,
CONSTRAINT [PK_delays] PRIMARY KEY CLUSTERED
([OriginCityName] ASC))
GO
```

8. When the `GO` statement is entered, the previous statements are evaluated.

The query creates a table named `delays`, which has a clustered index.

9. Use the following query to verify that the table is created:

```
SELECT * FROM information_schema.tables
GO
```

The output is similar to the following text:

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE
databaseName	dbo	delays	BASE TABLE

10. Enter `exit` at the `1>` prompt to exit the `tsql` utility.

## Export and load the data

In the previous sections, you copied the transformed data at the location

`abfs://<container-name>@<storage-account-name>.dfs.core.windows.net/tutorials/flighthdelays/output`. In this section, you use Sqoop to export the data from

`abfs://<container-name>@<storage-account-name>.dfs.core.windows.net/tutorials/flighthdelays/output` to the table

you created in the Azure SQL Database.

1. Use the following command to verify that Sqoop can see your SQL database:

```
sqoop list-databases --connect jdbc:sqlserver://<SERVER_NAME>.database.windows.net:1433 --username <ADMIN_LOGIN> --password <ADMIN_PASSWORD>
```

The command returns a list of databases, including the database in which you created the **delays** table.

2. Use the following command to export data from the **hivesamptable** table to the **delays** table:

```
sqoop export --connect 'jdbc:sqlserver://<SERVER_NAME>.database.windows.net:1433;database=<DATABASE_NAME>' --username <ADMIN_LOGIN> --password <ADMIN_PASSWORD> --table 'delays' --export-dir 'abfs://<container-name>@.dfs.core.windows.net/tutorials/flightdelays/output' --fields-terminated-by '\t' -m 1
```

Sqoop connects to the database that contains the **delays** table, and exports data from the **/tutorials/flightdelays/output** directory to the **delays** table.

3. After the `sqoop` command finishes, use the `tsql` utility to connect to the database:

```
TDSVER=8.0 tsql -H <SERVER_NAME>.database.windows.net -U <ADMIN_LOGIN> -P <ADMIN_PASSWORD> -p 1433 -D <DATABASE_NAME>
```

4. Use the following statements to verify that the data was exported to the **delays** table:

```
SELECT * FROM delays
GO
```

You should see a listing of data in the table. The table includes the city name and the average flight delay time for that city.

5. Enter `exit` to exit the `tsql` utility.

## Clean up resources

All resources used in this tutorial are preexisting. No cleanup is necessary.

## Next steps

To learn more ways to work with data in HDInsight, see the following article:

[Use Azure Data Lake Storage Gen2 with Azure HDInsight clusters](#)

# Tutorial: Implement the data lake capture pattern to update a Databricks Delta table

8/22/2022 • 10 minutes to read • [Edit Online](#)

This tutorial shows you how to handle events in a storage account that has a hierarchical namespace.

You'll build a small solution that enables a user to populate a Databricks Delta table by uploading a comma-separated values (csv) file that describes a sales order. You'll build this solution by connecting together an Event Grid subscription, an Azure Function, and a [Job](#) in Azure Databricks.

In this tutorial, you will:

- Create an Event Grid subscription that calls an Azure Function.
- Create an Azure Function that receives a notification from an event, and then runs the job in Azure Databricks.
- Create a Databricks job that inserts a customer order into a Databricks Delta table that is located in the storage account.

We'll build this solution in reverse order, starting with the Azure Databricks workspace.

## Prerequisites

- If you don't have an Azure subscription, create a [free account](#) before you begin.
- Create a storage account that has a hierarchical namespace (Azure Data Lake Storage Gen2). This tutorial uses a storage account named `contosoorders`. Make sure that your user account has the [Storage Blob Data Contributor role](#) assigned to it.

See [Create a storage account to use with Azure Data Lake Storage Gen2](#).

- Create a service principal. See [How to: Use the portal to create an Azure AD application and service principal that can access resources](#).

There's a couple of specific things that you'll have to do as you perform the steps in that article.

✓ When performing the steps in the [Assign the application to a role](#) section of the article, make sure to assign the [Storage Blob Data Contributor role](#) to the service principal.

### IMPORTANT

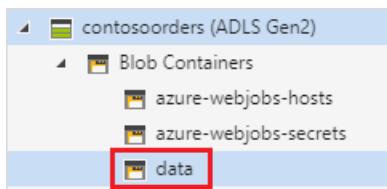
Make sure to assign the role in the scope of the Data Lake Storage Gen2 storage account. You can assign a role to the parent resource group or subscription, but you'll receive permissions-related errors until those role assignments propagate to the storage account.

✓ When performing the steps in the [Get values for signing in](#) section of the article, paste the tenant ID, app ID, and password values into a text file. You'll need those values later.

## Create a sales order

First, create a csv file that describes a sales order, and then upload that file to the storage account. Later, you'll use the data from this file to populate the first row in our Databricks Delta table.

1. Open Azure Storage Explorer. Then, navigate to your storage account, and in the **Blob Containers** section, create a new container named **data**.



For more information about how to use Storage Explorer, see [Use Azure Storage Explorer to manage data in an Azure Data Lake Storage Gen2 account](#).

2. In the **data** container, create a folder named **input**.

3. Paste the following text into a text editor.

```
InvoiceNo,StockCode,Description,Quantity,InvoiceDate,UnitPrice,CustomerID,Country
536365,85123A,WHITE HANGING HEART T-LIGHT HOLDER,6,12/1/2010 8:26,2.55,17850,United Kingdom
```

4. Save this file to your local computer and give it the name **data.csv**.

5. In Storage Explorer, upload this file to the **input** folder.

## Create a job in Azure Databricks

In this section, you'll perform these tasks:

- Create an Azure Databricks workspace.
- Create a notebook.
- Create and populate a Databricks Delta table.
- Add code that inserts rows into the Databricks Delta table.
- Create a Job.

### Create an Azure Databricks workspace

In this section, you create an Azure Databricks workspace using the Azure portal.

1. In the Azure portal, select **Create a resource > Analytics > Azure Databricks**.

Home >

## New

 Search the Marketplace

Azure Marketplace [See all](#)

Get started

Recently created

AI + Machine Learning

**Analytics**

Blockchain

Compute

Containers

Databases

Developer Tools

DevOps

Identity

Integration

Internet of Things

Media

Mixed Reality

Featured [See all](#)



Azure Data Explorer

[Learn more](#)



Azure HDInsight

[Quickstarts + tutorials](#)



Data Lake Analytics

[Quickstarts + tutorials](#)



Stream Analytics job

[Quickstarts + tutorials](#)



Analysis Services

[Quickstarts + tutorials](#)



Azure Databricks

[Quickstarts + tutorials](#)



Power BI Embedded

[Quickstarts + tutorials](#)

- Under **Azure Databricks Service**, provide the values to create a Databricks workspace.

Home > New > Azure Databricks Service

## Azure Databricks Service

\* Workspace name  
contoso-orders ✓

\* Subscription ⓘ  
contoso

\* Resource group ⓘ  
 Create new  Use existing  
contoso

\* Location  
West US

\* Pricing Tier ( [View full pricing details](#) )  
Standard (Apache Spark, Secure with Azur...)

Deploy Azure Databricks workspace in your Virtual Network ([preview](#))  
 Yes  No

[Create](#) [Automation options](#)

The workspace creation takes a few minutes. To monitor the operation status, view the progress bar at the top.

### Create a Spark cluster in Databricks

1. In the [Azure portal](#), go to the Azure Databricks workspace that you created, and then select **Launch Workspace**.
2. You are redirected to the Azure Databricks portal. From the portal, select **New > Cluster**.

The screenshot shows the Azure Databricks portal interface. At the top, there's a logo and the text "Azure Databricks". Below it, a section titled "Featured Notebooks" displays three items: "Introduction to Apache Spark on Databricks" (Python icon), "Databricks for Data Scientists" (red cube icon), and "Introduction to Structured Streaming" (Python icon). The main area is divided into three columns: "New" (with "Cluster" highlighted in red), "Documentation" (with links to "Databricks Guide", "Python, R, Scala, SQL", and "Importing Data"), and "Open Recent" (with a note about recent files and a link to the "welcome guide").

3. In the **New cluster** page, provide the values to create a cluster.

**Create Cluster**

## New Cluster

Cancel **Create Cluster** 2-8 Workers: 28.0-112.0 GB Memory, 8-32 Cores, 0.75 DBU

Cluster Name: customer-order-cluster

Cluster Mode: Standard

Pool: None

Databricks Runtime Version: Runtime: 5.3 (Scala 2.11, Spark 2.4.1) Learn more

Python Version: 3

Autopilot Options:  Enable autoscaling  Terminate after 120 minutes of inactivity

Worker Type: Standard\_DS3\_v2 (14.0 GB Memory, 4 Cores, 0.75 DBU) Min Workers: 2 Max Workers: 8

Driver Type: Same as worker (14.0 GB Memory, 4 Cores, 0.75 DBU)

**Advanced Options**

Accept all other default values other than the following:

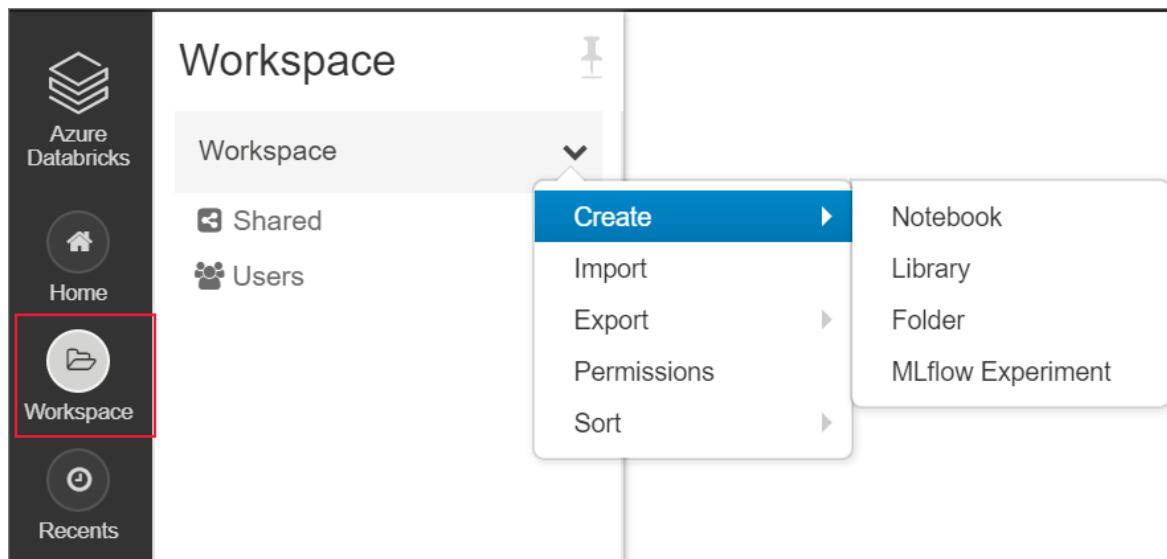
- Enter a name for the cluster.
- Make sure you select the **Terminate after 120 minutes of inactivity** checkbox. Provide a duration (in minutes) to terminate the cluster, if the cluster is not being used.

4. Select **Create cluster**. Once the cluster is running, you can attach notebooks to the cluster and run Spark jobs.

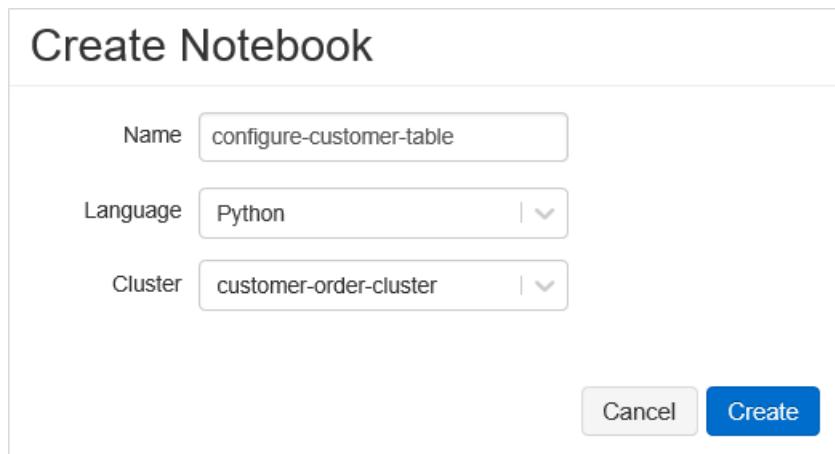
For more information on creating clusters, see [Create a Spark cluster in Azure Databricks](#).

### Create a notebook

1. In the left pane, select **Workspace**. From the **Workspace** drop-down, select **Create > Notebook**.



2. In the **Create Notebook** dialog box, enter a name for the notebook. Select **Python** as the language, and then select the Spark cluster that you created earlier.



Select **Create**.

### Create and populate a Databricks Delta table

1. In the notebook that you created, copy and paste the following code block into the first cell, but don't run this code yet.

Replace the `appId`, `password`, `tenant` placeholder values in this code block with the values that you collected while completing the prerequisites of this tutorial.

```
dbutils.widgets.text('source_file', "", "Source File")

spark.conf.set("fs.azure.account.auth.type", "OAuth")
spark.conf.set("fs.azure.account.oauth.provider.type",
"org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider")
spark.conf.set("fs.azure.account.oauth2.client.id", "<appId>")
spark.conf.set("fs.azure.account.oauth2.client.secret", "<password>")
spark.conf.set("fs.azure.account.oauth2.client.endpoint",
"https://login.microsoftonline.com/<tenant>/oauth2/token")

adlsPath = 'abfss://data@contosoorders.dfs.core.windows.net/'
inputPath = adlsPath + dbutils.widgets.get('source_file')
customerTablePath = adlsPath + 'delta-tables/customers'
```

This code creates a widget named `source_file`. Later, you'll create an Azure Function that calls this code and passes a file path to that widget. This code also authenticates your service principal with the storage account, and creates some variables that you'll use in other cells.

#### NOTE

In a production setting, consider storing your authentication key in Azure Databricks. Then, add a look up key to your code block instead of the authentication key.

For example, instead of using this line of code:

`spark.conf.set("fs.azure.account.oauth2.client.secret", "<password>")`, you would use the following line of code:

```
spark.conf.set("fs.azure.account.oauth2.client.secret", dbutils.secrets.get(scope = "<scope-name>", key = "<key-name-for-service-credential>"))
```

After you've completed this tutorial, see the [Azure Data Lake Storage Gen2](#) article on the Azure Databricks Website to see examples of this approach.

2. Press the **SHIFT + ENTER** keys to run the code in this block.
3. Copy and paste the following code block into a different cell, and then press the **SHIFT + ENTER** keys to run the code in this block.

```
from pyspark.sql.types import StructType, StructField, DoubleType, IntegerType, StringType

inputSchema = StructType([
 StructField("InvoiceNo", IntegerType(), True),
 StructField("StockCode", StringType(), True),
 StructField("Description", StringType(), True),
 StructField("Quantity", IntegerType(), True),
 StructField("InvoiceDate", StringType(), True),
 StructField("UnitPrice", DoubleType(), True),
 StructField("CustomerID", IntegerType(), True),
 StructField("Country", StringType(), True)
])

rawDataDF = (spark.read
 .option("header", "true")
 .schema(inputSchema)
 .csv(adlsPath + 'input')
)

(rawDataDF.write
 .mode("overwrite")
 .format("delta")
 .saveAsTable("customer_data", path=customerTablePath))
```

This code creates the Databricks Delta table in your storage account, and then loads some initial data from the csv file that you uploaded earlier.

4. After this code block successfully runs, remove this code block from your notebook.

#### Add code that inserts rows into the Databricks Delta table

1. Copy and paste the following code block into a different cell, but don't run this cell.

```
upsertDataDF = (spark
 .read
 .option("header", "true")
 .csv(inputPath)
)
upsertDataDF.createOrReplaceTempView("customer_data_to_upsert")
```

This code inserts data into a temporary table view by using data from a csv file. The path to that csv file comes from the input widget that you created in an earlier step.

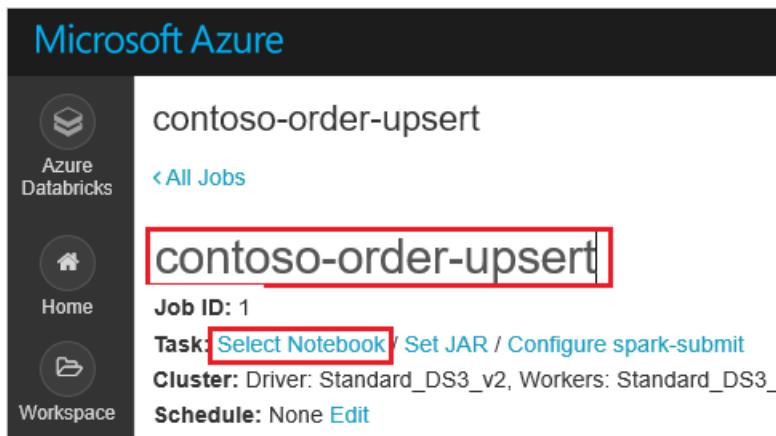
2. Add the following code to merge the contents of the temporary table view with the Databricks Delta table.

```
%sql
MERGE INTO customer_data cd
USING customer_data_to_upsert cu
ON cd.CustomerID = cu.CustomerID
WHEN MATCHED THEN
 UPDATE SET
 cd.StockCode = cu.StockCode,
 cd.Description = cu.Description,
 cd.InvoiceNo = cu.InvoiceNo,
 cd.Quantity = cu.Quantity,
 cd.InvoiceDate = cu.InvoiceDate,
 cd.UnitPrice = cu.UnitPrice,
 cd.Country = cu.Country
WHEN NOT MATCHED
 THEN INSERT (InvoiceNo, StockCode, Description, Quantity, InvoiceDate, UnitPrice, CustomerID, Country)
VALUES (
 cu.InvoiceNo,
 cu.StockCode,
 cu.Description,
 cu.Quantity,
 cu.InvoiceDate,
 cu.UnitPrice,
 cu.CustomerID,
 cu.Country)
```

## Create a Job

Create a Job that runs the notebook that you created earlier. Later, you'll create an Azure Function that runs this job when an event is raised.

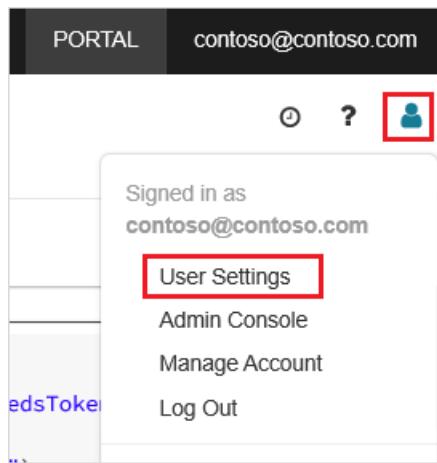
1. Click **Jobs**.
2. In the **Jobs** page, click **Create Job**.
3. Give the job a name, and then choose the `upsert-order-data` workbook.



## Create an Azure Function

Create an Azure Function that runs the Job.

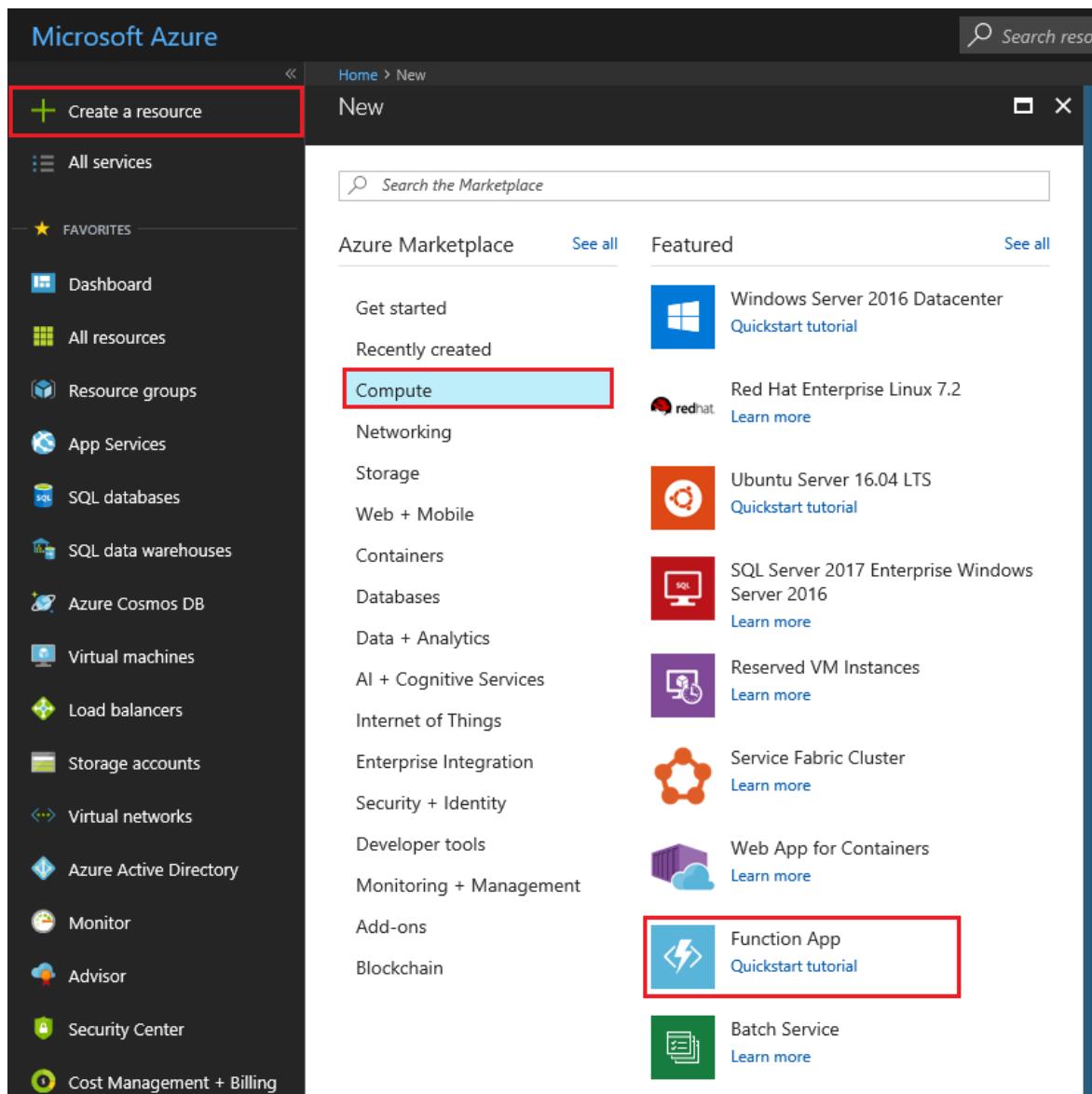
1. In the upper corner of the Databricks workspace, choose the people icon, and then choose **User settings**.



2. Click the **Generate new token** button, and then click the **Generate** button.

Make sure to copy the token to safe place. Your Azure Function needs this token to authenticate with Databricks so that it can run the Job.

3. Select the **Create a resource** button found on the upper left corner of the Azure portal, then select **Compute > Function App**.



4. In the **Create** page of the Function App, make sure to select **.NET Core** for the runtime stack, and make sure to configure an Application Insights instance.

**Function App**

Create

\* App name  
contosoorder .azurewebsites.net

\* Subscription  
contoso

\* Resource Group ⓘ  
 Create new  Use existing  
contoso

\* OS  
 Windows  Linux

\* Hosting Plan ⓘ  
Consumption Plan

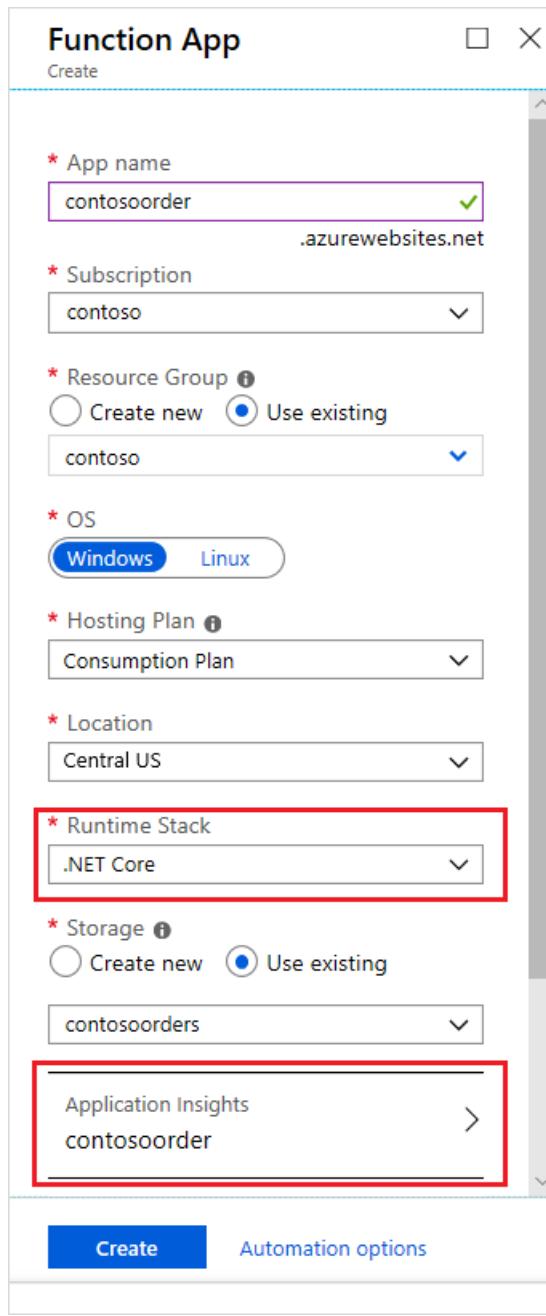
\* Location  
Central US

\* Runtime Stack  
.NET Core

\* Storage ⓘ  
 Create new  Use existing  
contosoorders

Application Insights  
contosoorder

**Create** Automation options



5. In the **Overview** page of the Function App, click **Configuration**.

The screenshot shows the Azure Function Apps overview page for the function app 'contsoorder'. The 'Overview' tab is active. Key details shown include:

- Status:** Running
- Subscription:** contoso
- Subscription ID:** 66775-22222-4183-a31c-5205094

In the 'Configured features' section, the 'Configuration' link is highlighted with a red box.

6. In the Application Settings page, choose the New application setting button to add each setting.

The screenshot shows the 'Application settings' page. The 'Application settings' tab is selected. A red box highlights the 'New application setting' button.

Add the following settings:

SETTING NAME	VALUE
DBX_INSTANCE	The region of your databricks workspace. For example: westus2.azuredatabricks.net
DBX_PAT	The personal access token that you generated earlier.
DBX_JOB_ID	The identifier of the running job. In our case, this value is 1.

7. In the overview page of the function app, click the New function button.

The screenshot shows the Azure Functions blade in the Azure portal. The left sidebar has a search bar with "contsoorder" and a delete button. Below it are sections for "Function Apps", "contsoorder" (expanded), "Functions" (selected, highlighted in blue), "Proxies", and "Slots (preview)". The main area is titled "Functions" with a search bar. A table with columns "NAME" and "STATUS" shows "No results". At the top right of the main area, there is a red box around the "New function" button.

8. Choose **Azure Event Grid Trigger**.

Install the `Microsoft.Azure.WebJobs.Extensions.EventGrid` extension if you're prompted to do so. If you have to install it, then you'll have to choose **Azure Event Grid Trigger** again to create the function.

The **New Function** pane appears.

9. In the **New Function** pane, name the function **UpsertOrder**, and then click the **Create** button.

10. Replace the contents of the code file with this code, and then click the **Save** button:

```

using "Microsoft.Azure.EventGrid"
using "Newtonsoft.Json"
using Microsoft.Azure.EventGrid.Models;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;

private static HttpClient httpClient = new HttpClient();

public static async Task Run(EventGridEvent eventGridEvent, ILogger log)
{
 log.LogInformation("Event Subject: " + eventGridEvent.Subject);
 log.LogInformation("Event Topic: " + eventGridEvent.Topic);
 log.LogInformation("Event Type: " + eventGridEvent.EventType);
 log.LogInformation(eventGridEvent.Data.ToString());

 if (eventGridEvent.EventType == "Microsoft.Storage.BlobCreated" || eventGridEvent.EventType == "Microsoft.Storage.FileRenamed") {
 var fileData = ((JObject)(eventGridEvent.Data)).ToObject<StorageBlobCreatedEventData>();
 if (fileData.Api == "FlushWithClose") {
 log.LogInformation("Triggering Databricks Job for file: " + fileData.Url);
 var fileUrl = new Uri(fileData.Url);
 var httpRequestMessage = new HttpRequestMessage {
 Method = HttpMethod.Post,
 RequestUri = new Uri(String.Format("https:///{0}/api/2.0/jobs/run-now",
System.Environment.GetEnvironmentVariable("DBX_INSTANCE", EnvironmentVariableTarget.Process))),
 Headers = {
 { System.Net.HttpRequestHeader.Authorization.ToString(), "Bearer " +
System.Environment.GetEnvironmentVariable ("DBX_PAT", EnvironmentVariableTarget.Process)},
 { System.Net.HttpRequestHeader.ContentType.ToString (), "application/json" }
 },
 Content = new StringContent(JsonConvert.SerializeObject(new {
 job_id = System.Environment.GetEnvironmentVariable ("DBX_JOB_ID",
EnvironmentVariableTarget.Process) ,
 notebook_params = new {
 source_file = String.Join("", fileUrl.Segments.Skip(2))
 }
 })),
 };
 var response = await httpClient.SendAsync(httpRequestMessage);
 response.EnsureSuccessStatusCode();
 }
 }
}

```

This code parses information about the storage event that was raised, and then creates a request message with url of the file that triggered the event. As part of the message, the function passes a value to the **source\_file** widget that you created earlier. the function code sends the message to the Databricks Job and uses the token that you obtained earlier as authentication.

## Create an Event Grid subscription

In this section, you'll create an Event Grid subscription that calls the Azure Function when files are uploaded to the storage account.

1. In the function code page, click the **Add Event Grid subscription** button.

```

1 #r "Microsoft.Azure.EventGrid"
2 #r "Newtonsoft.Json"
3 using Microsoft.Azure.EventGrid.Models;
4 using Newtonsoft.Json;
5 using Newtonsoft.Json.Linq;
6

```

2. In the **Create Event Subscription** page, name the subscription, and then use the fields in the page to select your storage account.

**EVENT SUBSCRIPTION DETAILS**

Name: contoso-order-event-subscription

Event Schema: Event Grid Schema

**TOPIC DETAILS**

Topic Types: Storage Accounts

Subscription: contoso

Resource Group: contoso

Resource: contosoorders

**EVENT TYPES**

Filter to Event Types: 2 selected

**ENDPOINT DETAILS**

Create

3. In the **Filter to Event Types** drop-down list, select the **Blob Created**, and **Blob Deleted** events, and then click the **Create** button.

## Test the Event Grid subscription

1. Create a file named `customer-order.csv`, paste the following information into that file, and save it to your local computer.

```

InvoiceNo,StockCode,Description,Quantity,InvoiceDate,UnitPrice,CustomerID,Country
536371,99999,EverGlow Single,228,1/1/2018 9:01,33.85,20993,Sierra Leone

```

2. In Storage Explorer, upload this file to the input folder of your storage account.

Uploading a file raises the **Microsoft.Storage.BlobCreated** event. Event Grid notifies all subscribers to that event. In our case, the Azure Function is the only subscriber. The Azure Function parses the event parameters to determine which event occurred. It then passes the URL of the file to the Databricks Job. The Databricks Job reads the file, and adds a row to the Databricks Delta table that is located in your storage account.

account.

3. To check if the job succeeded, open your databricks workspace, click the **Jobs** button, and then open your job.
4. Select the job to open the job page.

The screenshot shows the Microsoft Azure Databricks Jobs page. On the left, there's a sidebar with icons for Azure Databricks, Home, and Workspace. The main area is titled "Jobs" and has a "Create Job" button. A table lists a single job entry:

Name ↑	Job ID	Created By	Task
contoso-order-upsert	1	consto employee	upsert-order-data

When the job completes, you'll see a completion status.

The screenshot shows a table of completed jobs in the past 60 days. The table includes columns for Run, Run ID, Start Time, Launched, Duration, Spark, and Status. One row is highlighted with a red box around the "Status" column, which shows "Succeeded".

Run	Run ID	Start Time	Launched	Duration	Spark	Status
Run 10	10	2019-08-12 14:44:44 PDT	Manually	5m 34s	Spark UI / Logs / Metrics	Succeeded

5. In a new workbook cell, run this query in a cell to see the updated delta table.

```
%sql select * from customer_data
```

The returned table shows the latest record.

The screenshot shows a Databricks notebook cell with a command line input and a table output. The command is "%sql select \* from customer\_data". The table output shows two rows of data. The second row, which contains the latest record, is highlighted with a red box.

InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850	United Kingdom
536371	99999	EverGlow Single	228	1/1/2018 9:01	33.85	20993	Sierra Leone

6. To update this record, create a file named `customer-order-update.csv`, paste the following information into that file, and save it to your local computer.

```
InvoiceNo,StockCode,Description,Quantity,InvoiceDate,UnitPrice,CustomerID,Country
536371,99999,EverGlow Single,22,1/1/2018 9:01,33.85,20993,Sierra Leone
```

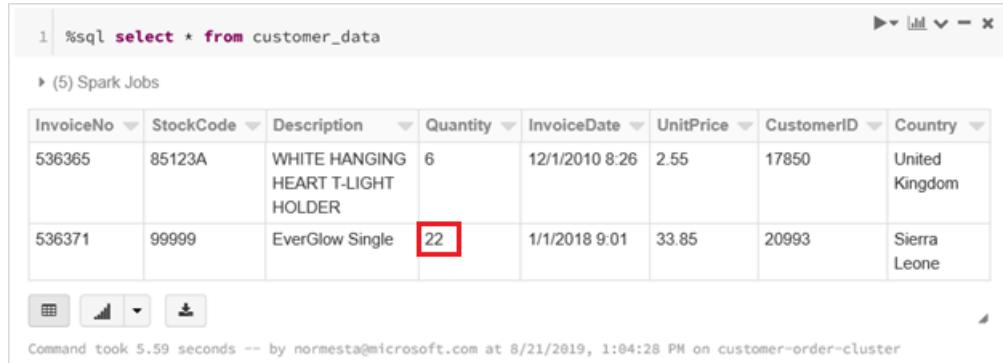
This csv file is almost identical to the previous one except the quantity of the order is changed from `228` to `22`.

7. In Storage Explorer, upload this file to the **input** folder of your storage account.

8. Run the `select` query again to see the updated delta table.

```
%sql select * from customer_data
```

The returned table shows the updated record.



InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850	United Kingdom
536371	99999	EverGlow Single	22	1/1/2018 9:01	33.85	20993	Sierra Leone

Command took 5.59 seconds -- by normesta@microsoft.com at 8/21/2019, 1:04:28 PM on customer-order-cluster

## Clean up resources

When they're no longer needed, delete the resource group and all related resources. To do so, select the resource group for the storage account and select **Delete**.

## Next steps

[Reacting to Blob storage events](#)

# Tutorials that use Azure services with Azure Data Lake Storage Gen2

8/22/2022 • 2 minutes to read • [Edit Online](#)

This article contains links to tutorials that show you how to use various Azure services with Data Lake Storage Gen2.

## List of tutorials

AZURE SERVICE	STEP-BY-STEP GUIDE
Azure Synapse Analytics	<a href="#">Get Started with Azure Synapse Analytics</a>
Azure Data Factory	<a href="#">Load data into Azure Data Lake Storage Gen2 with Azure Data Factory</a>
Azure Databricks	<a href="#">Use with Azure Databricks</a>
Azure Databricks	<a href="#">Extract, transform, and load data by using Azure Databricks</a>
Azure Databricks	<a href="#">Access Data Lake Storage Gen2 data with Azure Databricks using Spark</a>
Azure Event Grid	<a href="#">Implement the data lake capture pattern to update a Databricks Delta table</a>
Azure Machine Learning	<a href="#">Access data in Azure storage services</a>
Azure Data Box	<a href="#">Use Azure Data Box to migrate data from an on-premises HDFS store to Azure Storage</a>
HDInsight	<a href="#">Use Azure Data Lake Storage Gen2 with Azure HDInsight clusters</a>
HDInsight	<a href="#">Extract, transform, and load data by using Apache Hive on Azure HDInsight</a>
Power BI	<a href="#">Analyze data in Data Lake Storage Gen2 using Power BI</a>
Azure Data Explorer	<a href="#">Query data in Azure Data Lake using Azure Data Explorer</a>
Azure Cognitive Search	<a href="#">Index and search Azure Data Lake Storage Gen2 documents (preview)</a>

### NOTE

This table doesn't reflect the complete list of Azure services that support Data Lake Storage Gen2. To see a list of supported Azure services, their level of support, see [Azure services that support Azure Data Lake Storage Gen2](#). To see how services organized into categories such as ingest, download, process, and visualize, see [Ingest, process, and analyze](#).

## See also

[Best practices for using Azure Data Lake Storage Gen2](#)

# Best practices for using Azure Data Lake Storage Gen2

8/22/2022 • 14 minutes to read • [Edit Online](#)

This article provides best practice guidelines that help you optimize performance, reduce costs, and secure your Data Lake Storage Gen2 enabled Azure Storage account.

For general suggestions around structuring a data lake, see these articles:

- [Overview of Azure Data Lake Storage for the data management and analytics scenario](#)
- [Provision three Azure Data Lake Storage Gen2 accounts for each data landing zone](#)

## Find documentation

Azure Data Lake Storage Gen2 is not a dedicated service or account type. It's a set of capabilities that support high throughput analytic workloads. The Data Lake Storage Gen2 documentation provides best practices and guidance for using these capabilities. Refer to the [Blob storage documentation](#) content, for all other aspects of account management such as setting up network security, designing for high availability, and disaster recovery.

### Evaluate feature support and known issues

Use the following pattern as you configure your account to use Blob storage features.

1. Review the [Blob Storage feature support in Azure Storage accounts](#) article to determine whether a feature is fully supported in your account. Some features aren't yet supported or have partial support in Data Lake Storage Gen2 enabled accounts. Feature support is always expanding so make sure to periodically review this article for updates.
2. Review the [Known issues with Azure Data Lake Storage Gen2](#) article to see if there are any limitations or special guidance around the feature you intend to use.
3. Scan feature articles for any guidance that is specific to Data Lake Storage Gen2 enabled accounts.

### Understand the terms used in documentation

As you move between content sets, you'll notice some slight terminology differences. For example, content featured in the [Blob storage documentation](#), will use the term *blob* instead of *file*. Technically, the files that you ingest to your storage account become blobs in your account. Therefore, the term is correct. However, this can cause confusion if you're use to the term *file*. You'll also see the term *container* used to refer to a *file system*. Consider these terms as synonymous.

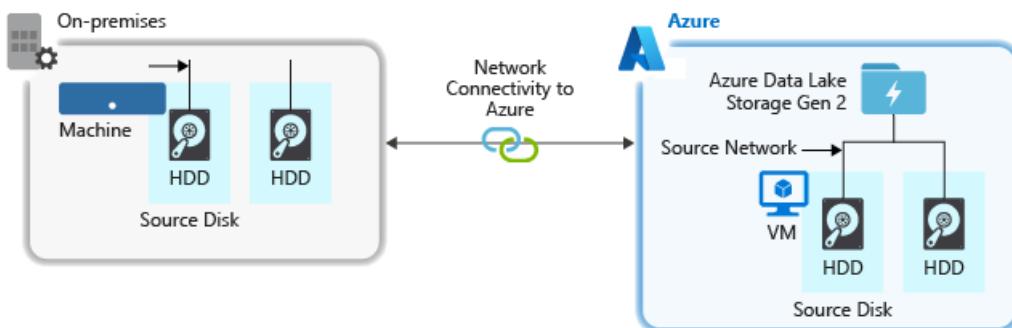
## Consider premium

If your workloads require a low consistent latency and/or require a high number of input output operations per second (IOP), consider using a premium block blob storage account. This type of account makes data available via high-performance hardware. Data is stored on solid-state drives (SSDs) which are optimized for low latency. SSDs provide higher throughput compared to traditional hard drives. The storage costs of premium performance are higher, but transaction costs are lower, so if your workloads execute a large number of transactions, a premium performance block blob account can be economical.

If your storage account is going to be used for analytics, we highly recommend that you use Azure Data Lake Storage Gen2 along with a premium block blob storage account. This combination of using premium block blob storage accounts along with a Data Lake Storage enabled account is referred to as the [premium tier for Azure Data Lake Storage](#).

# Optimize for data ingest

When ingesting data from a source system, the source hardware, source network hardware, or the network connectivity to your storage account can be a bottleneck.



## Source hardware

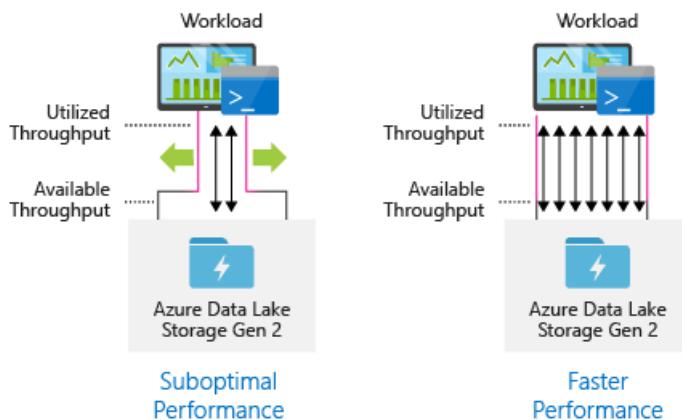
Whether you are using on-premises machines or Virtual Machines (VMs) in Azure, make sure to carefully select the appropriate hardware. For disk hardware, consider using Solid State Drives (SSD) and pick disk hardware that has faster spindles. For network hardware, use the fastest Network Interface Controllers (NIC) as possible. On Azure, we recommend Azure D14 VMs, which have the appropriately powerful disk and networking hardware.

## Network connectivity to the storage account

The network connectivity between your source data and your storage account can sometimes be a bottleneck. When your source data is on premise, consider using a dedicated link with [Azure ExpressRoute](#). If your source data is in Azure, the performance is best when the data is in the same Azure region as your Data Lake Storage Gen2 enabled account.

## Configure data ingestion tools for maximum parallelization

To achieve the best performance, use all available throughput by performing as many reads and writes in parallel as possible.



The following table summarizes the key settings for several popular ingestion tools.

TOOL	SETTINGS
DistCp	-m (mapper)
Azure Data Factory	parallelCopies
Sqoop	fs.azure.block.size, -m (mapper)

#### **NOTE**

The overall performance of your ingest operations depend on other factors that are specific to the tool that you're using to ingest data. For the best up-to-date guidance, see the documentation for each tool that you intend to use.

Your account can scale to provide the necessary throughput for all analytics scenarios. By default, a Data Lake Storage Gen2 enabled account provides enough throughput in its default configuration to meet the needs of a broad category of use cases. If you run into the default limit, the account can be configured to provide more throughput by contacting [Azure Support](#).

## Structure data sets

Consider pre-planning the structure of your data. File format, file size, and directory structure can all impact performance and cost.

### **File formats**

Data can be ingested in various formats. Data can appear in human readable formats such as JSON, CSV, or XML or as compressed binary formats such as `.tar.gz`. Data can come in various sizes as well. Data can be composed of large files (a few terabytes) such as data from an export of a SQL table from your on-premises systems. Data can also come in the form of a large number of tiny files (a few kilobytes) such as data from real-time events from an Internet of things (IoT) solution. You can optimize efficiency and costs by choosing an appropriate file format and file size.

Hadoop supports a set of file formats that are optimized for storing and processing structured data. Some common formats are Avro, Parquet, and Optimized Row Columnar (ORC) format. All of these formats are machine-readable binary file formats. They are compressed to help you manage file size. They have a schema embedded in each file, which makes them self-describing. The difference between these formats is in how data is stored. Avro stores data in a row-based format and the Parquet and ORC formats store data in a columnar format.

Consider using the Avro file format in cases where your I/O patterns are more write heavy, or the query patterns favor retrieving multiple rows of records in their entirety. For example, the Avro format works well with a message bus such as Event Hub or Kafka that write multiple events/messages in succession.

Consider Parquet and ORC file formats when the I/O patterns are more read heavy or when the query patterns are focused on a subset of columns in the records. Read transactions can be optimized to retrieve specific columns instead of reading the entire record.

Apache Parquet is an open source file format that is optimized for read heavy analytics pipelines. The columnar storage structure of Parquet lets you skip over non-relevant data. Your queries are much more efficient because they can narrowly scope which data to send from storage to the analytics engine. Also, because similar data types (for a column) are stored together, Parquet supports efficient data compression and encoding schemes that can lower data storage costs. Services such as [Azure Synapse Analytics](#), [Azure Databricks](#) and [Azure Data Factory](#) have native functionality that take advantage of Parquet file formats.

### **File size**

Larger files lead to better performance and reduced costs.

Typically, analytics engines such as HDInsight have a per-file overhead that involves tasks such as listing, checking access, and performing various metadata operations. If you store your data as many small files, this can negatively affect performance. In general, organize your data into larger sized files for better performance (256 MB to 100 GB in size). Some engines and applications might have trouble efficiently processing files that are greater than 100 GB in size.

Increasing file size can also reduce transaction costs. Read and write operations are billed in 4 megabyte

increments so you're charged for operation whether or not the file contains 4 megabytes or only a few kilobytes. For pricing information, see [Azure Data Lake Storage pricing](#).

Sometimes, data pipelines have limited control over the raw data, which has lots of small files. In general, we recommend that your system have some sort of process to aggregate small files into larger ones for use by downstream applications. If you're processing data in real time, you can use a real time streaming engine (such as [Azure Stream Analytics](#) or [Spark Streaming](#)) together with a message broker (such as [Event Hub](#) or [Apache Kafka](#)) to store your data as larger files. As you aggregate small files into larger ones, consider saving them in a read-optimized format such as [Apache Parquet](#) for downstream processing.

## Directory structure

Every workload has different requirements on how the data is consumed, but these are some common layouts to consider when working with Internet of Things (IoT), batch scenarios or when optimizing for time-series data.

### IoT structure

In IoT workloads, there can be a great deal of data being ingested that spans across numerous products, devices, organizations, and customers. It's important to pre-plan the directory layout for organization, security, and efficient processing of the data for down-stream consumers. A general template to consider might be the following layout:

```
{Region}/{SubjectMatter(s)}/{yyyy}/{mm}/{dd}/{hh}/
```

For example, landing telemetry for an airplane engine within the UK might look like the following structure:

```
UK/Planes/BA1293/Engine1/2017/08/11/12/
```

In this example, by putting the date at the end of the directory structure, you can use ACLs to more easily secure regions and subject matters to specific users and groups. If you put the data structure at the beginning, it would be much more difficult to secure these regions and subject matters. For example, if you wanted to provide access only to UK data or certain planes, you'd need to apply a separate permission for numerous directories under every hour directory. This structure would also exponentially increase the number of directories as time went on.

### Batch jobs structure

A commonly used approach in batch processing is to place data into an "in" directory. Then, once the data is processed, put the new data into an "out" directory for downstream processes to consume. This directory structure is sometimes used for jobs that require processing on individual files, and might not require massively parallel processing over large datasets. Like the IoT structure recommended above, a good directory structure has the parent-level directories for things such as region and subject matters (for example, organization, product, or producer). Consider date and time in the structure to allow better organization, filtered searches, security, and automation in the processing. The level of granularity for the date structure is determined by the interval on which the data is uploaded or processed, such as hourly, daily, or even monthly.

Sometimes file processing is unsuccessful due to data corruption or unexpected formats. In such cases, a directory structure might benefit from a /bad folder to move the files to for further inspection. The batch job might also handle the reporting or notification of these *bad* files for manual intervention. Consider the following template structure:

```
{Region}/{SubjectMatter(s)}/In/{yyyy}/{mm}/{dd}/{hh}/\\
{Region}/{SubjectMatter(s)}/Out/{yyyy}/{mm}/{dd}/{hh}/\\
{Region}/{SubjectMatter(s)}/Bad/{yyyy}/{mm}/{dd}/{hh}/
```

For example, a marketing firm receives daily data extracts of customer updates from their clients in North America. It might look like the following snippet before and after being processed:

```
NA/Extracts/ACMEPaperCo/In/2017/08/14/updates_08142017.csv\\
NA/Extracts/ACMEPaperCo/Out/2017/08/14/processed_updates_08142017.csv
```

In the common case of batch data being processed directly into databases such as Hive or traditional SQL databases, there isn't a need for an `/in` or `/out` directory because the output already goes into a separate folder for the Hive table or external database. For example, daily extracts from customers would land into their respective directories. Then, a service such as [Azure Data Factory](#), [Apache Oozie](#), or [Apache Airflow](#) would trigger a daily Hive or Spark job to process and write the data into a Hive table.

#### Time series data structure

For Hive workloads, partition pruning of time-series data can help some queries read only a subset of the data, which improves performance.

Those pipelines that ingest time-series data, often place their files with a structured naming for files and folders. Below is a common example we see for data that is structured by date:

```
\DataSet\YYYY\MM\DD\datafile_YYYY_MM_DD.tsv
```

Notice that the datetime information appears both as folders and in the filename.

For date and time, the following is a common pattern

```
\DataSet\YYYY\MM\DD\HH\mm\datafile_YYYY_MM_DD_HH_mm.tsv
```

Again, the choice you make with the folder and file organization should optimize for the larger file sizes and a reasonable number of files in each folder.

## Set up security

Start by reviewing the recommendations in the [Security recommendations for Blob storage](#) article. You'll find best practice guidance about how to protect your data from accidental or malicious deletion, secure data behind a firewall, and use Azure Active Directory (Azure AD) as the basis of identity management.

Then, review the [Access control model in Azure Data Lake Storage Gen2](#) article for guidance that is specific to Data Lake Storage Gen2 enabled accounts. This article helps you understand how to use Azure role-based access control (Azure RBAC) roles together with access control lists (ACLs) to enforce security permissions on directories and files in your hierarchical file system.

## Ingest, process, and analyze

There are many different sources of data and different ways in which that data can be ingested into a Data Lake Storage Gen2 enabled account.

For example, you can ingest large sets of data from HDInsight and Hadoop clusters or smaller sets of *ad hoc* data for prototyping applications. You can ingest streamed data that is generated by various sources such as applications, devices, and sensors. For this type of data, you can use tools to capture and process the data on an event-by-event basis in real time, and then write the events in batches into your account. You can also ingest web server logs which contain information such as the history of page requests. For log data, consider writing custom scripts or applications to upload them so that you'll have the flexibility to include your data uploading component as part of your larger big data application.

Once the data is available in your account, you can run analysis on that data, create visualizations, and even download data to your local machine or to other repositories such as an Azure SQL database or SQL Server instance.

The following table recommends tools that you can use to ingest, analyze, visualize, and download data. Use the links in this table to find guidance about how to configure and use each tool.

PURPOSE	TOOLS & TOOL GUIDANCE
Ingest ad hoc data	Azure portal, <a href="#">Azure PowerShell</a> , <a href="#">Azure CLI</a> , <a href="#">REST</a> , <a href="#">Azure Storage Explorer</a> , <a href="#">Apache DistCp</a> , <a href="#">AzCopy</a>
Ingest relational data	<a href="#">Azure Data Factory</a>
Ingest web server logs	<a href="#">Azure PowerShell</a> , <a href="#">Azure CLI</a> , <a href="#">REST</a> , Azure SDKs (.NET, Java, Python, and Node.js), <a href="#">Azure Data Factory</a>
Ingest from HDInsight clusters	<a href="#">Azure Data Factory</a> , <a href="#">Apache DistCp</a> , <a href="#">AzCopy</a>
Ingest from Hadoop clusters	<a href="#">Azure Data Factory</a> , <a href="#">Apache DistCp</a> , <a href="#">WANdisco LiveData Migrator for Azure</a> , <a href="#">Azure Data Box</a>
Ingest large data sets (several terabytes)	<a href="#">Azure ExpressRoute</a>
Process & analyze data	<a href="#">Azure Synapse Analytics</a> , <a href="#">Azure HDInsight</a> , <a href="#">Databricks</a>
Visualize data	<a href="#">Power BI</a> , <a href="#">Azure Data Lake Storage query acceleration</a>
Download data	Azure portal, <a href="#">PowerShell</a> , <a href="#">Azure CLI</a> , <a href="#">REST</a> , Azure SDKs (.NET, Java, Python, and Node.js), <a href="#">Azure Storage Explorer</a> , <a href="#">AzCopy</a> , <a href="#">Azure Data Factory</a> , <a href="#">Apache DistCp</a>

#### NOTE

This table doesn't reflect the complete list of Azure services that support Data Lake Storage Gen2. To see a list of supported Azure services, their level of support, see [Azure services that support Azure Data Lake Storage Gen2](#).

## Monitor telemetry

Monitoring use and performance is an important part of operationalizing your service. Examples include frequent operations, operations with high latency, or operations that cause service-side throttling.

All of the telemetry for your storage account is available through [Azure Storage logs in Azure Monitor](#). This feature integrates your storage account with Log Analytics and Event Hubs, while also enabling you to archive logs to another storage account. To see the full list of metrics and resources logs and their associated schema, see [Azure Storage monitoring data reference](#).

Where you choose to store your logs depends on how you plan to access them. For example, if you want to access your logs in near real time, and be able to correlate events in logs with other metrics from Azure Monitor, you can store your logs in a Log Analytics workspace. This allows you to query your logs using KQL and author queries, which enumerate the `StorageBlobLogs` table in your workspace.

If you want to store your logs for both near real-time query and long term retention, you can configure your diagnostic settings to send logs to both a Log Analytics workspace and a storage account.

If you want to access your logs through another query engine such as Splunk, you can configure your diagnostic settings to send logs to an Event Hub and ingest logs from the Event Hub to your chosen destination.

Azure Storage logs in Azure Monitor can be enabled through the Azure portal, PowerShell, the Azure CLI, and Azure Resource Manager templates. For at-scale deployments, Azure Policy can be used with full support for remediation tasks. For more information, see [Azure/Community-Policy](#) and [ciphertxt/AzureStoragePolicy](#).

## See also

- [Key considerations for Azure Data Lake Storage](#)
- [Access control model in Azure Data Lake Storage Gen2](#)
- [The hitchhiker's guide to the Data Lake](#)
- [Overview of Azure Data Lake Storage Gen2](#)

# Azure Data Lake Storage query acceleration

8/22/2022 • 4 minutes to read • [Edit Online](#)

Query acceleration enables applications and analytics frameworks to dramatically optimize data processing by retrieving only the data that they require to perform a given operation. This reduces the time and processing power that is required to gain critical insights into stored data.

## Overview

Query acceleration accepts filtering *predicates* and *column projections* which enable applications to filter rows and columns at the time that data is read from disk. Only the data that meets the conditions of a predicate are transferred over the network to the application. This reduces network latency and compute cost.

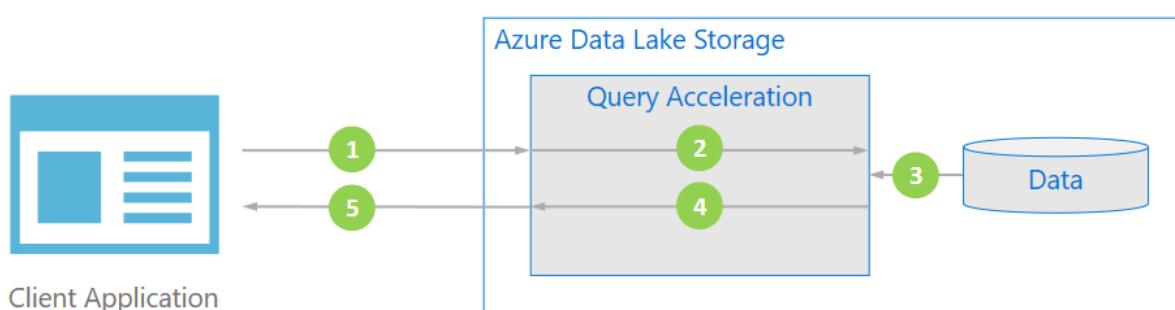
You can use SQL to specify the row filter predicates and column projections in a query acceleration request. A request processes only one file. Therefore, advanced relational features of SQL, such as joins and group by aggregates, aren't supported. Query acceleration supports CSV and JSON formatted data as input to each request.

The query acceleration feature isn't limited to Data Lake Storage (storage accounts that have the hierarchical namespace enabled on them). Query acceleration is completely compatible with the blobs in storage accounts that **don't** have a hierarchical namespace enabled on them. This means that you can achieve the same reduction in network latency and compute costs when you process data that you already have stored as blobs in storage accounts.

For an example of how to use query acceleration in a client application, see [Filter data by using Azure Data Lake Storage query acceleration](#).

## Data flow

The following diagram illustrates how a typical application uses query acceleration to process data.



1. The client application requests file data by specifying predicates and column projections.
2. Query acceleration parses the specified SQL query and distributes work to parse and filter data.
3. Processors read the data from the disk, parses the data by using the appropriate format, and then filters data by applying the specified predicates and column projections.
4. Query acceleration combines the response shards to stream back to client application.
5. The client application receives and parses the streamed response. The application doesn't need to filter any additional data and can apply the desired calculation or transformation directly.

## Better performance at a lower cost

Query acceleration optimizes performance by reducing the amount of data that gets transferred and processed by your application.

To calculate an aggregated value, applications commonly retrieve **all** of the data from a file, and then process and filter the data locally. An analysis of the input/output patterns for analytics workloads reveal that applications typically require only 20% of the data that they read to perform any given calculation. This statistic is true even after applying techniques such as [partition pruning](#). This means that 80% of that data is needlessly transferred across the network, parsed, and filtered by applications. This pattern, essentially designed to remove unneeded data, incurs a significant compute cost.

Even though Azure features an industry-leading network, in terms of both throughput and latency, needlessly transferring data across that network is still costly for application performance. By filtering out the unwanted data during the storage request, query acceleration eliminates this cost.

Additionally, the CPU load that is required to parse and filter unneeded data requires your application to provision a greater number and larger VMs in order to do its work. By transferring this compute load to query acceleration, applications can realize significant cost savings.

## Applications that can benefit from query acceleration

Query acceleration is designed for distributed analytics frameworks and data processing applications.

Distributed analytics frameworks such as Apache Spark and Apache Hive, include a storage abstraction layer within the framework. These engines also include query optimizers that can incorporate knowledge of the underlying I/O service's capabilities when determining an optimal query plan for user queries. These frameworks are beginning to integrate query acceleration. As a result, users of these frameworks will see improved query latency and a lower total cost of ownership without having to make any changes to the queries.

Query acceleration is also designed for data processing applications. These types of applications typically perform large-scale data transformations that might not directly lead to analytics insights so they don't always use established distributed analytics frameworks. These applications often have a more direct relationship with the underlying storage service so they can benefit directly from features such as query acceleration.

For an example of how an application can integrate query acceleration, see [Filter data by using Azure Data Lake Storage query acceleration](#).

## Pricing

Due to the increased compute load within the Azure Data Lake Storage service, the pricing model for using query acceleration differs from the normal Azure Data Lake Storage transaction model. Query acceleration charges a cost for the amount of data scanned as well as a cost for the amount of data returned to the caller. For more information, see [Azure Data Lake Storage Gen2 pricing](#).

Despite the change to the billing model, Query acceleration's pricing model is designed to lower the total cost of ownership for a workload, given the reduction in the much more expensive VM costs.

## Next steps

- [Filter data by using Azure Data Lake Storage query acceleration](#)
- [Query acceleration SQL language reference](#)

# Premium tier for Azure Data Lake Storage

8/22/2022 • 2 minutes to read • [Edit Online](#)

Azure Data Lake Storage Gen2 now supports [premium block blob storage accounts](#). Premium block blob storage accounts are ideal for big data analytics applications and workloads that require low consistent latency and have a high number of transactions. Example workloads include interactive workloads, IoT, streaming analytics, artificial intelligence, and machine learning.

## TIP

To learn more about the performance and cost advantages of using a premium block blob storage account, and to see how other Data Lake Storage Gen2 customers have used this type of account, see [Premium block blob storage accounts](#).

## Getting started with premium

First, check to make sure your favorite Blob Storage features are compatible with premium block blob storage accounts, then create the account.

## NOTE

You can't convert an existing standard general-purpose v2 storage account to a premium block blob storage account. To migrate to a premium block blob storage account, you must create a premium block blob storage account, and migrate the data to the new account.

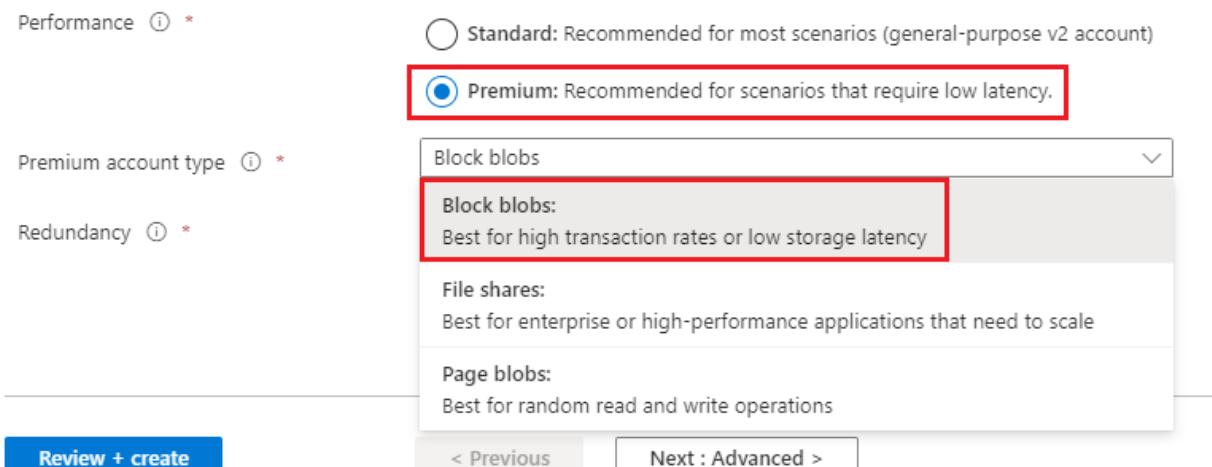
### Check for Blob Storage feature compatibility

Some Blob Storage features aren't yet supported or have partial support in premium block blob storage accounts. Before choosing premium, review the [Blob Storage feature support in Azure Storage accounts](#) article to determine whether the features that you intend to use are fully supported in your account. Feature support is always expanding so make sure to periodically review this article for updates.

### Create a new Storage account

Create a new Azure Storage account. For complete guidance, see [Create a storage account](#) account.

As you create the account, choose the **Premium** performance option and the **Block blobs** account type.



To unlock Azure Data Lake Storage Gen2 capabilities, enable the **Hierarchical namespace** setting in the

Advanced tab of the Create storage account page.

The following image shows this setting in the **Create storage account** page.

The screenshot shows the Microsoft Azure 'Create a storage account' interface. At the top, there's a navigation bar with 'Microsoft Azure', a search bar, and several icons. Below it, the breadcrumb navigation shows 'Home > Create a resource > Create a storage account'. The main title is 'Create a storage account' with a '...' button. Below the title, there are tabs: 'Basics' (disabled), 'Advanced' (highlighted with a red box), 'Networking', 'Data protection', 'Tags', and 'Review + create'. A note below the tabs says: 'Certain options have been disabled by default due to the combination of storage account performance, redundancy, and region.' Under the 'Security' section, there are five settings with checkboxes: 'Enable secure transfer' (checked), 'Enable infrastructure encryption' (unchecked), 'Enable blob public access' (checked), 'Enable storage account key access' (checked), and 'Minimum TLS version' set to 'Version 1.2'. In the 'Data Lake Storage Gen2' section, there's a note about hierarchical namespaces and a checkbox for 'Enable hierarchical namespace' (checked). The 'Enable hierarchical namespace' checkbox is also highlighted with a red box.

## Next steps

Use the premium tier for Azure Data Lake Storage with your favorite analytics service such as Azure Databricks, Azure HDInsight and Azure Synapse Analytics. See [Tutorials that use Azure services with Azure Data Lake Storage Gen2](#).

# The Azure Blob Filesystem driver (ABFS): A dedicated Azure Storage driver for Hadoop

8/22/2022 • 3 minutes to read • [Edit Online](#)

One of the primary access methods for data in Azure Data Lake Storage Gen2 is via the [Hadoop FileSystem](#). Data Lake Storage Gen2 allows users of Azure Blob Storage access to a new driver, the Azure Blob File System driver or [ABFS](#). ABFS is part of Apache Hadoop and is included in many of the commercial distributions of Hadoop. Using this driver, many applications and frameworks can access data in Azure Blob Storage without any code explicitly referencing Data Lake Storage Gen2.

## Prior capability: The Windows Azure Storage Blob driver

The Windows Azure Storage Blob driver or [WASB driver](#) provided the original support for Azure Blob Storage. This driver performed the complex task of mapping file system semantics (as required by the Hadoop FileSystem interface) to that of the object store style interface exposed by Azure Blob Storage. This driver continues to support this model, providing high performance access to data stored in blobs, but contains a significant amount of code performing this mapping, making it difficult to maintain. Additionally, some operations such as [FileSystem.rename\(\)](#) and [FileSystem.delete\(\)](#) when applied to directories require the driver to perform a vast number of operations (due to object stores lack of support for directories) which often leads to degraded performance. The ABFS driver was designed to overcome the inherent deficiencies of WASB.

## The Azure Blob File System driver

The [Azure Data Lake Storage REST interface](#) is designed to support file system semantics over Azure Blob Storage. Given that the Hadoop FileSystem is also designed to support the same semantics there is no requirement for a complex mapping in the driver. Thus, the Azure Blob File System driver (or ABFS) is a mere client shim for the REST API.

However, there are some functions that the driver must still perform:

### URI scheme to reference data

Consistent with other FileSystem implementations within Hadoop, the ABFS driver defines its own URI scheme so that resources (directories and files) may be distinctly addressed. The URI scheme is documented in [Use the Azure Data Lake Storage Gen2 URI](#). The structure of the URI is:

```
abfs[s]://file_system@account_name.dfs.core.windows.net/<path>/<path>/<file_name>
```

Using the above URI format, standard Hadoop tools and frameworks can be used to reference these resources:

```
hdfs dfs -mkdir -p abfs://fileanalysis@myanalytics.dfs.core.windows.net/tutorials/flightdelays/data
hdfs dfs -put flight_delays.csv
abfs://fileanalysis@myanalytics.dfs.core.windows.net/tutorials/flightdelays/data/
```

Internally, the ABFS driver translates the resource(s) specified in the URI to files and directories and makes calls to the Azure Data Lake Storage REST API with those references.

### Authentication

The ABFS driver supports two forms of authentication so that the Hadoop application may securely access resources contained within a Data Lake Storage Gen2 capable account. Full details of the available authentication schemes are provided in the [Azure Storage security guide](#). They are:

- **Shared Key:** This permits users access to ALL resources in the account. The key is encrypted and stored in Hadoop configuration.
- **Azure Active Directory OAuth Bearer Token:** Azure AD bearer tokens are acquired and refreshed by the driver using either the identity of the end user or a configured Service Principal. Using this authentication model, all access is authorized on a per-call basis using the identity associated with the supplied token and evaluated against the assigned POSIX Access Control List (ACL).

**NOTE**

Azure Data Lake Storage Gen2 supports only Azure AD v1.0 endpoints.

## Configuration

All configuration for the ABFS driver is stored in the `core-site.xml` configuration file. On Hadoop distributions featuring [Ambari](#), the configuration may also be managed using the web portal or Ambari REST API.

Details of all supported configuration entries are specified in the [Official Hadoop documentation](#).

## Hadoop documentation

The ABFS driver is fully documented in the [Official Hadoop documentation](#)

## Next steps

- [Create an Azure Databricks Cluster](#)
- [Use the Azure Data Lake Storage Gen2 URI](#)

# Use the Azure Data Lake Storage Gen2 URI

8/22/2022 • 2 minutes to read • [Edit Online](#)

The [Hadoop Filesystem](#) driver that is compatible with Azure Data Lake Storage Gen2 is known by its scheme identifier `abfs` (Azure Blob File System). Consistent with other Hadoop Filesystem drivers, the ABFS driver employs a URI format to address files and directories within a Data Lake Storage Gen2 capable account.

## URI syntax

The URI syntax for Data Lake Storage Gen2 is dependent on whether or not your storage account is set up to have Data Lake Storage Gen2 as the default file system.

If the Data Lake Storage Gen2 capable account you wish to address is **not** set as the default file system during account creation, then the shorthand URI syntax is:

```
abfs[s]1://<file_system>2@<account_name>3.dfs.core.windows.net/<path>4/<file_name>5
```

- 1. Scheme identifier:** The `abfs` protocol is used as the scheme identifier. If you add an 's' at the end (abfss) then the ABFS Hadoop client driver will *ALWAYS* use Transport Layer Security (TLS) irrespective of the authentication method chosen. If you choose OAuth as your authentication then the client driver will always use TLS even if you specify 'abfs' instead of 'abfss' because OAuth solely relies on the TLS layer. Finally, if you choose to use the older method of storage account key, then the client driver will interpret 'abfs' to mean that you do not want to use TLS.
- 2. File system:** The parent location that holds the files and folders. This is the same as Containers in the Azure Storage Blobs service.
- 3. Account name:** The name given to your storage account during creation.
- 4. Paths:** A forward slash delimited (`/`) representation of the directory structure.
- 5. File name:** The name of the individual file. This parameter is optional if you are addressing a directory.

However, if the account you wish to address is set as the default file system during account creation, then the shorthand URI syntax is:

```
/<path>1/<file_name>2
```

- 1. Path:** A forward slash delimited (`/`) representation of the directory structure.
- 2. File Name:** The name of the individual file.

## Next steps

- [Use Azure Data Lake Storage Gen2 with Azure HDInsight clusters](#)

# Azure Data Lake Storage Gen2 hierarchical namespace

8/22/2022 • 3 minutes to read • [Edit Online](#)

A key mechanism that allows Azure Data Lake Storage Gen2 to provide file system performance at object storage scale and prices is the addition of a **hierarchical namespace**. This allows the collection of objects/files within an account to be organized into a hierarchy of directories and nested subdirectories in the same way that the file system on your computer is organized. With a hierarchical namespace enabled, a storage account becomes capable of providing the scalability and cost-effectiveness of object storage, with file system semantics that are familiar to analytics engines and frameworks.

## The benefits of a hierarchical namespace

The following benefits are associated with file systems that implement a hierarchical namespace over blob data:

- **Atomic directory manipulation:** Object stores approximate a directory hierarchy by adopting a convention of embedding slashes (/) in the object name to denote path segments. While this convention works for organizing objects, the convention provides no assistance for actions like moving, renaming or deleting directories. Without real directories, applications must process potentially millions of individual blobs to achieve directory-level tasks. By contrast, a hierarchical namespace processes these tasks by updating a single entry (the parent directory).

This dramatic optimization is especially significant for many big data analytics frameworks. Tools like Hive, Spark, etc. often write output to temporary locations and then rename the location at the conclusion of the job. Without a hierarchical namespace, this rename can often take longer than the analytics process itself. Lower job latency equals lower total cost of ownership (TCO) for analytics workloads.

- **Familiar Interface Style:** File systems are well understood by developers and users alike. There is no need to learn a new storage paradigm when you move to the cloud as the file system interface exposed by Data Lake Storage Gen2 is the same paradigm used by computers, large and small.

One of the reasons that object stores haven't historically supported a hierarchical namespace is that a hierarchical namespace limits scale. However, the Data Lake Storage Gen2 hierarchical namespace scales linearly and does not degrade either the data capacity or performance.

## Deciding whether to enable a hierarchical namespace

After you've enabled a hierarchical namespace on your account, you can't revert it back to a flat namespace. Therefore, consider whether it makes sense to enable a hierarchical namespace based on the nature of your object store workloads. To evaluate the impact of enabling a hierarchical namespace on workloads, applications, costs, service integrations, tools, features, and documentation, see [Upgrading Azure Blob Storage with Azure Data Lake Storage Gen2 capabilities](#).

Some workloads might not gain any benefit by enabling a hierarchical namespace. Examples include backups, image storage, and other applications where object organization is stored separately from the objects themselves (for example: in a separate database).

Also, while support for Blob storage features and the Azure service ecosystem continues to grow, there are still some features and Azure services that are not yet supported in accounts that have a hierarchical namespace. See [Known Issues](#).

In general, we recommend that you turn on a hierarchical namespace for storage workloads that are designed for file systems that manipulate directories. This includes all workloads that are primarily for analytics processing. Datasets that require a high degree of organization will also benefit by enabling a hierarchical namespace.

The reasons for enabling a hierarchical namespace are determined by a TCO analysis. Generally speaking, improvements in workload latency due to storage acceleration will require compute resources for less time. Latency for many workloads may be improved due to atomic directory manipulation that is enabled by a hierarchical namespace. In many workloads, the compute resource represents > 85% of the total cost and so even a modest reduction in workload latency equates to a significant amount of TCO savings. Even in cases where enabling a hierarchical namespace increases storage costs, the TCO is still lowered due to reduced compute costs.

To analyze differences in data storage prices, transaction prices, and storage capacity reservation pricing between accounts that have a flat hierarchical namespace versus a hierarchical namespace, see [Azure Data Lake Storage Gen2 pricing](#).

## Next steps

- Enable a hierarchical namespace when you create a new storage account. See [Create a Storage account](#).
- Enable a hierarchical namespace on an existing storage account. See [Upgrade Azure Blob Storage with Azure Data Lake Storage Gen2 capabilities](#).

# Security recommendations for Blob storage

8/22/2022 • 9 minutes to read • [Edit Online](#)

This article contains security recommendations for Blob storage. Implementing these recommendations will help you fulfill your security obligations as described in our shared responsibility model. For more information on how Microsoft fulfills service provider responsibilities, see [Shared responsibility in the cloud](#).

Some of the recommendations included in this article can be automatically monitored by Microsoft Defender for Cloud, which is the first line of defense in protecting your resources in Azure. For information on Microsoft Defender for Cloud, see [What is Microsoft Defender for Cloud?](#)

Microsoft Defender for Cloud periodically analyzes the security state of your Azure resources to identify potential security vulnerabilities. It then provides you with recommendations on how to address them. For more information on Microsoft Defender for Cloud recommendations, see [Review your security recommendations](#).

## Data protection

RECOMMENDATION	COMMENTS	DEFENDER FOR CLOUD
Use the Azure Resource Manager deployment model	Create new storage accounts using the Azure Resource Manager deployment model for important security enhancements, including superior Azure role-based access control (Azure RBAC) and auditing, Resource Manager-based deployment and governance, access to managed identities, access to Azure Key Vault for secrets, and Azure AD-based authentication and authorization for access to Azure Storage data and resources. If possible, migrate existing storage accounts that use the classic deployment model to use Azure Resource Manager. For more information about Azure Resource Manager, see <a href="#">Azure Resource Manager overview</a> .	-
Enable Microsoft Defender for all of your storage accounts	Microsoft Defender for Storage provides an additional layer of security intelligence that detects unusual and potentially harmful attempts to access or exploit storage accounts. Security alerts are triggered in Microsoft Defender for Cloud when anomalies in activity occur and are also sent via email to subscription administrators, with details of suspicious activity and recommendations on how to investigate and remediate threats. For more information, see <a href="#">Configure Microsoft Defender for Storage</a> .	Yes

RECOMMENDATION	COMMENTS	DEFENDER FOR CLOUD
Turn on soft delete for blobs	Soft delete for blobs enables you to recover blob data after it has been deleted. For more information on soft delete for blobs, see <a href="#">Soft delete for Azure Storage blobs</a> .	-
Turn on soft delete for containers	Soft delete for containers enables you to recover a container after it has been deleted. For more information on soft delete for containers, see <a href="#">Soft delete for containers</a> .	-
Lock storage account to prevent accidental or malicious deletion or configuration changes	Apply an Azure Resource Manager lock to your storage account to protect the account from accidental or malicious deletion or configuration change. Locking a storage account does not prevent data within that account from being deleted. It only prevents the account itself from being deleted. For more information, see <a href="#">Apply an Azure Resource Manager lock to a storage account</a> .	-
Store business-critical data in immutable blobs	Configure legal holds and time-based retention policies to store blob data in a WORM (Write Once, Read Many) state. Blobs stored immutably can be read, but cannot be modified or deleted for the duration of the retention interval. For more information, see <a href="#">Store business-critical blob data with immutable storage</a> .	-
Require secure transfer (HTTPS) to the storage account	When you require secure transfer for a storage account, all requests to the storage account must be made over HTTPS. Any requests made over HTTP are rejected. Microsoft recommends that you always require secure transfer for all of your storage accounts. For more information, see <a href="#">Require secure transfer to ensure secure connections</a> .	-
Limit shared access signature (SAS) tokens to HTTPS connections only	Requiring HTTPS when a client uses a SAS token to access blob data helps to minimize the risk of eavesdropping. For more information, see <a href="#">Grant limited access to Azure Storage resources using shared access signatures (SAS)</a> .	-

## Identity and access management

RECOMMENDATION	COMMENTS	DEFENDER FOR CLOUD
Use Azure Active Directory (Azure AD) to authorize access to blob data	Azure AD provides superior security and ease of use over Shared Key for authorizing requests to Blob storage. For more information, see <a href="#">Authorize access to data in Azure Storage</a> .	-
Keep in mind the principal of least privilege when assigning permissions to an Azure AD security principal via Azure RBAC	When assigning a role to a user, group, or application, grant that security principal only those permissions that are necessary for them to perform their tasks. Limiting access to resources helps prevent both unintentional and malicious misuse of your data.	-
Use a user delegation SAS to grant limited access to blob data to clients	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 is analogous to a service SAS in terms of its scope and function, but offers security benefits over the service SAS. For more information, see <a href="#">Grant limited access to Azure Storage resources using shared access signatures (SAS)</a> .	-
Secure your account access keys with Azure Key Vault	Microsoft recommends using Azure AD to authorize requests to Azure Storage. However, if you must use Shared Key authorization, then secure your account keys with Azure Key Vault. You can retrieve the keys from the key vault at runtime, instead of saving them with your application. For more information about Azure Key Vault, see <a href="#">Azure Key Vault overview</a> .	-
Regenerate your account keys periodically	Rotating the account keys periodically reduces the risk of exposing your data to malicious actors.	-
Disallow Shared Key authorization	When you disallow Shared Key authorization for a storage account, Azure Storage rejects all subsequent requests to that account that are authorized with the account access keys. Only secured requests that are authorized with Azure AD will succeed. For more information, see <a href="#">Prevent Shared Key authorization for an Azure Storage account</a> .	-
Keep in mind the principal of least privilege when assigning permissions to a SAS	When creating a SAS, specify only those permissions that are required by the client to perform its function. Limiting access to resources helps prevent both unintentional and malicious misuse of your data.	-

RECOMMENDATION	COMMENTS	DEFENDER FOR CLOUD
Have a revocation plan in place for any SAS that you issue to clients	If a SAS is compromised, you will want to revoke that SAS as soon as possible. To revoke a user delegation SAS, revoke the user delegation key to quickly invalidate all signatures associated with that key. To revoke a service SAS that is associated with a stored access policy, you can delete the stored access policy, rename the policy, or change its expiry time to a time that is in the past. For more information, see <a href="#">Grant limited access to Azure Storage resources using shared access signatures (SAS)</a> .	-
If a service SAS is not associated with a stored access policy, then set the expiry time to one hour or less	A service SAS that is not associated with a stored access policy cannot be revoked. For this reason, limiting the expiry time so that the SAS is valid for one hour or less is recommended.	-
Disable anonymous public read access to containers and blobs	Anonymous public read access to a container and its blobs grants read-only access to those resources to any client. Avoid enabling public read access unless your scenario requires it. To learn how to disable anonymous public access for a storage account, see <a href="#">Configure anonymous public read access for containers and blobs</a> .	-

## Networking

RECOMMENDATION	COMMENTS	DEFENDER FOR CLOUD
Configure the minimum required version of Transport Layer Security (TLS) for a storage account.	Require that clients use a more secure version of TLS to make requests against an Azure Storage account by configuring the minimum version of TLS for that account. For more information, see <a href="#">Configure minimum required version of Transport Layer Security (TLS) for a storage account</a>	-
Enable the <b>Secure transfer required</b> option on all of your storage accounts	When you enable the <b>Secure transfer required</b> option, all requests made against the storage account must take place over secure connections. Any requests made over HTTP will fail. For more information, see <a href="#">Require secure transfer in Azure Storage</a> .	Yes

RECOMMENDATION	COMMENTS	DEFENDER FOR CLOUD
Enable firewall rules	<p>Configure firewall rules to limit access to your storage account to requests that originate from specified IP addresses or ranges, or from a list of subnets in an Azure Virtual Network (VNet). For more information about configuring firewall rules, see <a href="#">Configure Azure Storage firewalls and virtual networks</a>.</p>	-
Allow trusted Microsoft services to access the storage account	<p>Turning on firewall rules for your storage account blocks incoming requests for data by default, unless the requests originate from a service operating within an Azure Virtual Network (VNet) or from allowed public IP addresses. Requests that are blocked include those from other Azure services, from the Azure portal, from logging and metrics services, and so on. You can permit requests from other Azure services by adding an exception to allow trusted Microsoft services to access the storage account. For more information about adding an exception for trusted Microsoft services, see <a href="#">Configure Azure Storage firewalls and virtual networks</a>.</p>	-
Use private endpoints	<p>A private endpoint assigns a private IP address from your Azure Virtual Network (VNet) to the storage account. It secures all traffic between your VNet and the storage account over a private link. For more information about private endpoints, see <a href="#">Connect privately to a storage account using Azure Private Endpoint</a>.</p>	-
Use VNet service tags	<p>A service tag represents a group of IP address prefixes from a given Azure service. Microsoft manages the address prefixes encompassed by the service tag and automatically updates the service tag as addresses change. For more information about service tags supported by Azure Storage, see <a href="#">Azure service tags overview</a>. For a tutorial that shows how to use service tags to create outbound network rules, see <a href="#">Restrict access to PaaS resources</a>.</p>	-
Limit network access to specific networks	<p>Limiting network access to networks hosting clients requiring access reduces the exposure of your resources to network attacks.</p>	Yes

RECOMMENDATION	COMMENTS	DEFENDER FOR CLOUD
Configure network routing preference	You can configure network routing preference for your Azure storage account to specify how network traffic is routed to your account from clients over the Internet using the Microsoft global network or Internet routing. For more information, see <a href="#">Configure network routing preference for Azure Storage</a> .	-

## Logging/Monitoring

RECOMMENDATION	COMMENTS	DEFENDER FOR CLOUD
Track how requests are authorized	Enable Azure Storage logging to track how each request made against Azure Storage was authorized. The logs indicate whether a request was made anonymously, by using an OAuth 2.0 token, by using Shared Key, or by using a shared access signature (SAS). For more information, see <a href="#">Monitoring Azure Blob Storage with Azure Monitor</a> or <a href="#">Azure Storage analytics logging with Classic Monitoring</a> .	-
Set up alerts in Azure Monitor	Configure log alerts to evaluate resources logs at a set frequency and fire an alert based on the results. For more information, see <a href="#">Log alerts in Azure Monitor</a> .	-

## Next steps

- [Azure security documentation](#)
- [Secure development documentation](#).

# Access control model in Azure Data Lake Storage Gen2

8/22/2022 • 7 minutes to read • [Edit Online](#)

Data Lake Storage Gen2 supports the following authorization mechanisms:

- Shared Key authorization
- Shared access signature (SAS) authorization
- Role-based access control (Azure RBAC)
- Access control lists (ACL)

[Shared Key and SAS authorization](#) grants access to a user (or application) without requiring them to have an identity in Azure Active Directory (Azure AD). With these two forms of authentication, Azure RBAC and ACLs have no effect.

Azure RBAC and ACL both require the user (or application) to have an identity in Azure AD. Azure RBAC lets you grant "coarse-grain" access to storage account data, such as read or write access to **all** of the data in a storage account, while ACLs let you grant "fine-grained" access, such as write access to a specific directory or file.

This article focuses on Azure RBAC and ACLs, and how the system evaluates them together to make authorization decisions for storage account resources.

## Role-based access control (Azure RBAC)

Azure RBAC uses role assignments to apply sets of permissions to [security principals](#). A security principal is an object that represents a user, group, service principal, or managed identity that is defined in Azure Active Directory (AD). A permission set can give a security principal a "coarse-grain" level of access such as read or write access to **all** of the data in a storage account or **all** of the data in a container.

The following roles permit a security principal to access data in a storage account.

ROLE	DESCRIPTION
<a href="#">Storage Blob Data Owner</a>	Full access to Blob storage containers and data. This access permits the security principal to set the owner an item, and to modify the ACLs of all items.
<a href="#">Storage Blob Data Contributor</a>	Read, write, and delete access to Blob storage containers and blobs. This access does not permit the security principal to set the ownership of an item, but it can modify the ACL of items that are owned by the security principal.
<a href="#">Storage Blob Data Reader</a>	Read and list Blob storage containers and blobs.

Roles such as [Owner](#), [Contributor](#), [Reader](#), and [Storage Account Contributor](#) permit a security principal to manage a storage account, but do not provide access to the data within that account. However, these roles (excluding [Reader](#)) can obtain access to the storage keys, which can be used in various client tools to access the data.

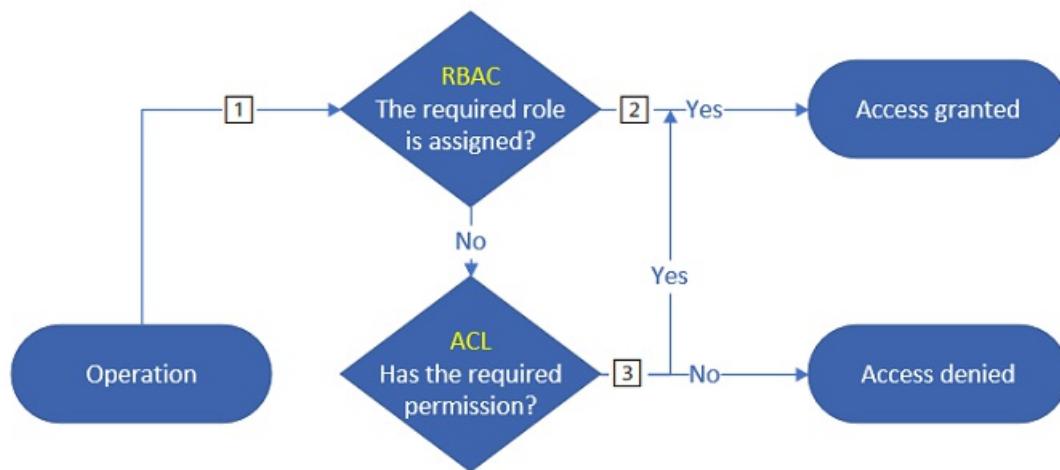
## Access control lists (ACLs)

ACLs give you the ability to apply "finer grain" level of access to directories and files. An *ACL* is a permission construct that contains a series of *ACL entries*. Each ACL entry associates security principal with an access level. To learn more, see [Access control lists \(ACLs\) in Azure Data Lake Storage Gen2](#).

## How permissions are evaluated

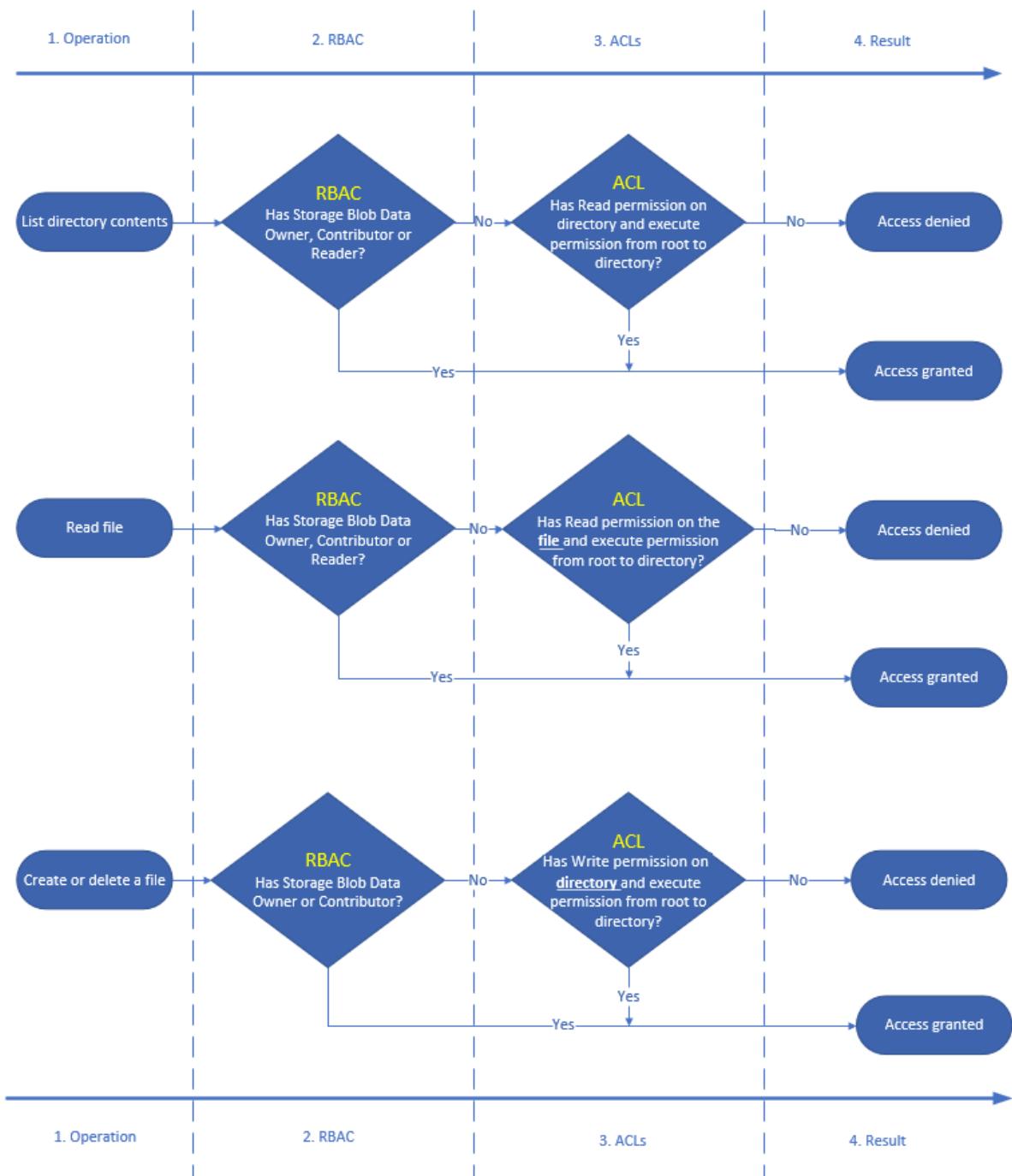
During security principal-based authorization, permissions are evaluated in the following order.

- 1 Azure role assignments are evaluated first and take priority over any ACL assignments.
- 2 If the operation is fully authorized based on Azure role assignment, then ACLs are not evaluated at all.
- 3 If the operation is not fully authorized, then ACLs are evaluated.



Because of the way that access permissions are evaluated by the system, you **cannot** use an ACL to **restrict** access that has already been granted by a role assignment. That's because the system evaluates Azure role assignments first, and if the assignment grants sufficient access permission, ACLs are ignored.

The following diagram shows the permission flow for three common operations: listing directory contents, reading a file, and writing a file.



## Permissions table: Combining Azure RBAC and ACL

The following table shows you how to combine Azure roles and ACL entries so that a security principal can perform the operations listed in the **Operation** column. This table shows a column for each level of a fictitious directory hierarchy. There's a column for the root directory of the container (/), a subdirectory named **Oregon**, a subdirectory of the Oregon directory named **Portland**, and a text file in the Portland directory named **Data.txt**. Appearing in those columns are [short form](#) representations of the ACL entry required to grant permissions. **N/A (Not applicable)** appears in the column if an ACL entry is not required to perform the operation.

OPERATION	ASSIGNED AZURE ROLE	/	OREGON/	PORLAND/	DATA.TXT
Read Data.txt	Storage Blob Data Owner	N/A	N/A	N/A	N/A

OPERATION	ASSIGNED AZURE ROLE	/	OREGON/	PORTLAND/	DATA.TXT
	Storage Blob Data Contributor	N/A	N/A	N/A	N/A
	Storage Blob Data Reader	N/A	N/A	N/A	N/A
	None	--X	--X	--X	R--
Append to Data.txt	Storage Blob Data Owner	N/A	N/A	N/A	N/A
	Storage Blob Data Contributor	N/A	N/A	N/A	N/A
	Storage Blob Data Reader	--X	--X	--X	-W-
	None	--X	--X	--X	RW-
Delete Data.txt	Storage Blob Data Owner	N/A	N/A	N/A	N/A
	Storage Blob Data Contributor	N/A	N/A	N/A	N/A
	Storage Blob Data Reader	--X	--X	-WX	N/A
	None	--X	--X	-WX	N/A
Create Data.txt	Storage Blob Data Owner	N/A	N/A	N/A	N/A
	Storage Blob Data Contributor	N/A	N/A	N/A	N/A
	Storage Blob Data Reader	--X	--X	-WX	N/A
	None	--X	--X	-WX	N/A
List /	Storage Blob Data Owner	N/A	N/A	N/A	N/A
	Storage Blob Data Contributor	N/A	N/A	N/A	N/A
	Storage Blob Data Reader	N/A	N/A	N/A	N/A
	None	R-X	N/A	N/A	N/A

OPERATION	ASSIGNED AZURE ROLE	/	OREGON/	PORLAND/	DATA.TXT
List /Oregon/	Storage Blob Data Owner	N/A	N/A	N/A	N/A
	Storage Blob Data Contributor	N/A	N/A	N/A	N/A
	Storage Blob Data Reader	N/A	N/A	N/A	N/A
	None	--X	R-X	N/A	N/A
List /Oregon/Portland/	Storage Blob Data Owner	N/A	N/A	N/A	N/A
	Storage Blob Data Contributor	N/A	N/A	N/A	N/A
	Storage Blob Data Reader	N/A	N/A	N/A	N/A
	None	--X	--X	R-X	N/A

#### NOTE

To view the contents of a container in Azure Storage Explorer, security principals must [sign in to Storage Explorer by using Azure AD](#), and (at a minimum) have read access (R--) to the root folder (\) of a container. This level of permission does give them the ability to list the contents of the root folder. If you don't want the contents of the root folder to be visible, you can assign them [Reader](#) role. With that role, they'll be able to list the containers in the account, but not container contents. You can then grant access to specific directories and files by using ACLs.

## Security groups

Always use [Azure AD security groups](#) as the assigned principal in an ACL entry. Resist the opportunity to directly assign individual users or service principals. Using this structure will allow you to add and remove users or service principals without the need to reapply ACLs to an entire directory structure. Instead, you can just add or remove users and service principals from the appropriate Azure AD security group.

There are many different ways to set up groups. For example, imagine that you have a directory named **/LogData** which holds log data that is generated by your server. Azure Data Factory (ADF) ingests data into that folder. Specific users from the service engineering team will upload logs and manage other users of this folder, and various Databricks clusters will analyze logs from that folder.

To enable these activities, you could create a **LogsWriter** group and a **LogsReader** group. Then, you could assign permissions as follows:

- Add the **LogsWriter** group to the ACL of the **/LogData** directory with **rwx** permissions.
- Add the **LogsReader** group to the ACL of the **/LogData** directory with **r-x** permissions.
- Add the service principal object or Managed Service Identity (MSI) for ADF to the **LogsWriters** group.
- Add users in the service engineering team to the **LogsWriter** group.
- Add the service principal object or MSI for Databricks to the **LogsReader** group.

If a user in the service engineering team leaves the company, you could just remove them from the [LogsWriter](#) group. If you did not add that user to a group, but instead, you added a dedicated ACL entry for that user, you would have to remove that ACL entry from the `/LogData` directory. You would also have to remove the entry from all subdirectories and files in the entire directory hierarchy of the `/LogData` directory.

To create a group and add members, see [Create a basic group and add members using Azure Active Directory](#).

## Limits on Azure role assignments and ACL entries

By using groups, you're less likely to exceed the maximum number of role assignments per subscription and the maximum number of ACL entries per file or directory. The following table describes these limits.

MECHANISM	SCOPE	LIMITS	SUPPORTED LEVEL OF PERMISSION
Azure RBAC	Storage accounts, containers. Cross resource Azure role assignments at subscription or resource group level.	2000 Azure role assignments in a subscription	Azure roles (built-in or custom)
ACL	Directory, file	32 ACL entries (effectively 28 ACL entries) per file and per directory. Access and default ACLs each have their own 32 ACL entry limit.	ACL permission

## Shared Key and Shared Access Signature (SAS) authorization

Azure Data Lake Storage Gen2 also supports [Shared Key](#) and [SAS](#) methods for authentication. A characteristic of these authentication methods is that no identity is associated with the caller and therefore security principal permission-based authorization cannot be performed.

In the case of Shared Key, the caller effectively gains 'super-user' access, meaning full access to all operations on all resources including data, setting owner, and changing ACLs.

SAS tokens include allowed permissions as part of the token. The permissions included in the SAS token are effectively applied to all authorization decisions, but no additional ACL checks are performed.

## Next steps

To learn more about access control lists, see [Access control lists \(ACLs\) in Azure Data Lake Storage Gen2](#).

# Access control lists (ACLs) in Azure Data Lake Storage Gen2

8/22/2022 • 16 minutes to read • [Edit Online](#)

Azure Data Lake Storage Gen2 implements an access control model that supports both Azure role-based access control (Azure RBAC) and POSIX-like access control lists (ACLs). This article describes access control lists in Data Lake Storage Gen2. To learn about how to incorporate Azure RBAC together with ACLs, and how system evaluates them to make authorization decisions, see [Access control model in Azure Data Lake Storage Gen2](#).

## About ACLs

You can associate a [security principal](#) with an access level for files and directories. Each association is captured as an entry in an *access control list (ACL)*. Each file and directory in your storage account has an access control list. When a security principal attempts an operation on a file or directory, An ACL check determines whether that security principal (user, group, service principal, or managed identity) has the correct permission level to perform the operation.

### NOTE

ACLs apply only to security principals in the same tenant, and they don't apply to users who use Shared Key or shared access signature (SAS) token authentication. That's because no identity is associated with the caller and therefore security principal permission-based authorization cannot be performed.

## How to set ACLs

To set file and directory level permissions, see any of the following articles:

ENVIRONMENT	ARTICLE
Azure Storage Explorer	<a href="#">Use Azure Storage Explorer to manage ACLs in Azure Data Lake Storage Gen2</a>
Azure portal	<a href="#">Use the Azure portal to manage ACLs in Azure Data Lake Storage Gen2</a>
.NET	<a href="#">Use .NET to manage ACLs in Azure Data Lake Storage Gen2</a>
Java	<a href="#">Use Java to manage ACLs in Azure Data Lake Storage Gen2</a>
Python	<a href="#">Use Python to manage ACLs in Azure Data Lake Storage Gen2</a>
JavaScript (Node.js)	<a href="#">Use the JavaScript SDK in Node.js to manage ACLs in Azure Data Lake Storage Gen2</a>
PowerShell	<a href="#">Use PowerShell to manage ACLs in Azure Data Lake Storage Gen2</a>

ENVIRONMENT	ARTICLE
Azure CLI	<a href="#">Use Azure CLI to manage ACLs in Azure Data Lake Storage Gen2</a>
REST API	<a href="#">Path - Update</a>

### IMPORTANT

If the security principal is a *service* principal, it's important to use the object ID of the service principal and not the object ID of the related app registration. To get the object ID of the service principal open the Azure CLI, and then use this command: `az ad sp show --id <Your App ID> --query objectId`. make sure to replace the `<Your App ID>` placeholder with the App ID of your app registration.

## Types of ACLs

There are two kinds of access control lists: *access ACLs* and *default ACLs*.

Access ACLs control access to an object. Files and directories both have access ACLs.

Default ACLs are templates of ACLs associated with a directory that determine the access ACLs for any child items that are created under that directory. Files do not have default ACLs.

Both access ACLs and default ACLs have the same structure.

### NOTE

Changing the default ACL on a parent does not affect the access ACL or default ACL of child items that already exist.

## Levels of permission

The permissions on directories and files in a container, are **Read**, **Write**, and **Execute**, and they can be used on files and directories as shown in the following table:

	FILE	DIRECTORY
<b>Read (R)</b>	Can read the contents of a file	Requires <b>Read</b> and <b>Execute</b> to list the contents of the directory
<b>Write (W)</b>	Can write or append to a file	Requires <b>Write</b> and <b>Execute</b> to create child items in a directory
<b>Execute (X)</b>	Does not mean anything in the context of Data Lake Storage Gen2	Required to traverse the child items of a directory

### NOTE

If you are granting permissions by using only ACLs (no Azure RBAC), then to grant a security principal read or write access to a file, you'll need to give the security principal **Execute** permissions to the root folder of the container, and to each folder in the hierarchy of folders that lead to the file.

## Short forms for permissions

RWX is used to indicate **Read + Write + Execute**. A more condensed numeric form exists in which **Read=4**,

**Write=2**, and **Execute=1**, the sum of which represents the permissions. Following are some examples.

NUMERIC FORM	SHORT FORM	WHAT IT MEANS
7	RWX	Read + Write + Execute
5	R-X	Read + Execute
4	R--	Read
0	---	No permissions

### Permissions inheritance

In the POSIX-style model that's used by Data Lake Storage Gen2, permissions for an item are stored on the item itself. In other words, permissions for an item cannot be inherited from the parent items if the permissions are set after the child item has already been created. Permissions are only inherited if default permissions have been set on the parent items before the child items have been created.

## Common scenarios related to ACL permissions

The following table shows you the ACL entries required to enable a security principal to perform the operations listed in the **Operation** column.

This table shows a column that represents each level of a fictitious directory hierarchy. There's a column for the root directory of the container (/), a subdirectory named **Oregon**, a subdirectory of the Oregon directory named **Portland**, and a text file in the Portland directory named **Data.txt**.

#### IMPORTANT

This table assumes that you are using **only** ACLs without any Azure role assignments. To see a similar table that combines Azure RBAC together with ACLs, see [Permissions table: Combining Azure RBAC and ACL](#).

OPERATION	/	OREGON/	PORTLAND/	DATA.TXT
Read Data.txt	--X	--X	--X	R--
Append to Data.txt	--X	--X	--X	RW-
Delete Data.txt	--X	--X	-WX	---
Create Data.txt	--X	--X	-WX	---
List /	R-X	---	---	---
List /Oregon/	--X	R-X	---	---
List /Oregon/Portland/	--X	--X	R-X	---

#### **NOTE**

Write permissions on the file are not required to delete it, so long as the previous two conditions are true.

## Users and identities

Every file and directory has distinct permissions for these identities:

- The owning user
- The owning group
- Named users
- Named groups
- Named service principals
- Named managed identities
- All other users

The identities of users and groups are Azure Active Directory (Azure AD) identities. So unless otherwise noted, a *user*, in the context of Data Lake Storage Gen2, can refer to an Azure AD user, service principal, managed identity, or security group.

### **The owning user**

The user who created the item is automatically the owning user of the item. An owning user can:

- Change the permissions of a file that is owned.
- Change the owning group of a file that is owned, as long as the owning user is also a member of the target group.

#### **NOTE**

The owning user *cannot* change the owning user of a file or directory. Only super-users can change the owning user of a file or directory.

### **The owning group**

In the POSIX ACLs, every user is associated with a *primary group*. For example, user "Alice" might belong to the "finance" group. Alice might also belong to multiple groups, but one group is always designated as their primary group. In POSIX, when Alice creates a file, the owning group of that file is set to her primary group, which in this case is "finance." The owning group otherwise behaves similarly to assigned permissions for other users/groups.

#### **Assigning the owning group for a new file or directory**

- **Case 1:** The root directory `/`. This directory is created when a Data Lake Storage Gen2 container is created. In this case, the owning group is set to the user who created the container if it was done using OAuth. If the container is created using Shared Key, an Account SAS, or a Service SAS, then the owner and owning group are set to `$superuser`.
- **Case 2 (every other case):** When a new item is created, the owning group is copied from the parent directory.

#### **Changing the owning group**

The owning group can be changed by:

- Any super-users.
- The owning user, if the owning user is also a member of the target group.

#### **NOTE**

The owning group cannot change the ACLs of a file or directory. While the owning group is set to the user who created the account in the case of the root directory, **Case 1** above, a single user account isn't valid for providing permissions via the owning group. You can assign this permission to a valid user group if applicable.

## How permissions are evaluated

Identities are evaluated in the following order:

1. Superuser
2. Owning user
3. Named user, service principal or managed identity
4. Owning group or named group
5. All other users

If more than one of these identities applies to a security principal, then the permission level associated with the first identity is granted. For example, if a security principal is both the owning user and a named user, then the permission level associated with the owning user applies.

The following pseudocode represents the access check algorithm for storage accounts. This algorithm shows the order in which identities are evaluated.

```

def access_check(user, desired_perms, path) :
 # access_check returns true if user has the desired permissions on the path, false otherwise
 # user is the identity that wants to perform an operation on path
 # desired_perms is a simple integer with values from 0 to 7 (R=4, W=2, X=1). User desires these
 permissions
 # path is the file or directory
 # Note: the "sticky bit" isn't illustrated in this algorithm

 # Handle super users.
 if (is_superuser(user)) :
 return True

 # Handle the owning user. Note that mask isn't used.
 entry = get_acl_entry(path, OWNER)
 if (user == entry.identity)
 return ((desired_perms & entry.permissions) == desired_perms)

 # Handle the named users. Note that mask IS used.
 entries = get_acl_entries(path, NAMED_USER)
 for entry in entries:
 if (user == entry.identity) :
 mask = get_mask(path)
 return ((desired_perms & entry.permissions & mask) == desired_perms)

 # Handle named groups and owning group
 member_count = 0
 perms = 0
 entries = get_acl_entries(path, NAMED_GROUP | OWNING_GROUP)
 mask = get_mask(path)
 for entry in entries:
 if (user_is_member_of_group(user, entry.identity)) :
 if ((desired_perms & entry.permissions & mask) == desired_perms)
 return True

 # Handle other
 perms = get_perms_for_other(path)
 mask = get_mask(path)
 return ((desired_perms & perms & mask) == desired_perms)

```

## The mask

As illustrated in the Access Check Algorithm, the mask limits access for named users, the owning group, and named groups.

For a new Data Lake Storage Gen2 container, the mask for the access ACL of the root directory ("/") defaults to **750** for directories and **640** for files. The following table shows the symbolic notation of these permission levels.

ENTITY	DIRECTORIES	FILES
Owning user	rwx	rwx
Owning group	r-x	r--
Other	---	---

Files do not receive the X bit as it is irrelevant to files in a store-only system.

The mask may be specified on a per-call basis. This allows different consuming systems, such as clusters, to have different effective masks for their file operations. If a mask is specified on a given request, it completely overrides the default mask.

## The sticky bit

The sticky bit is a more advanced feature of a POSIX container. In the context of Data Lake Storage Gen2, it is unlikely that the sticky bit will be needed. In summary, if the sticky bit is enabled on a directory, a child item can only be deleted or renamed by the child item's owning user, the directory's owner, or the Superuser (\$superuser).

The sticky bit isn't shown in the Azure portal.

## Default permissions on new files and directories

When a new file or directory is created under an existing directory, the default ACL on the parent directory determines:

- A child directory's default ACL and access ACL.
- A child file's access ACL (files do not have a default ACL).

### umask

When creating a file or directory, umask is used to modify how the default ACLs are set on the child item. umask is a 9-bit value on parent directories that contains an RWX value for **owning user**, **owning group**, and **other**.

The umask for Azure Data Lake Storage Gen2 a constant value that is set to 007. This value translates to:

UMASK COMPONENT	NUMERIC FORM	SHORT FORM	MEANING
umask.owning_user	0	---	For owning user, copy the parent's default ACL to the child's access ACL
umask.owning_group	0	---	For owning group, copy the parent's default ACL to the child's access ACL
umask.other	7	RWX	For other, remove all permissions on the child's access ACL

The umask value used by Azure Data Lake Storage Gen2 effectively means that the value for **other** is never transmitted by default on new children, unless a default ACL is defined on the parent directory. In that case, the umask is effectively ignored and the permissions defined by the default ACL are applied to the child item.

The following pseudocode shows how the umask is applied when creating the ACLs for a child item.

```
def set_default_acls_for_new_child(parent, child):
 child.acls = []
 for entry in parent.acls :
 new_entry = None
 if (entry.type == OWNING_USER) :
 new_entry = entry.clone(perms = entry.perms & (~umask.owning_user))
 elif (entry.type == OWNING_GROUP) :
 new_entry = entry.clone(perms = entry.perms & (~umask.owning_group))
 elif (entry.type == OTHER) :
 new_entry = entry.clone(perms = entry.perms & (~umask.other))
 else :
 new_entry = entry.clone(perms = entry.perms)
 child_acls.add(new_entry)
```

## FAQ

## Do I have to enable support for ACLs?

No. Access control via ACLs is enabled for a storage account as long as the Hierarchical Namespace (HNS) feature is turned ON.

If HNS is turned OFF, the Azure RBAC authorization rules still apply.

## What is the best way to apply ACLs?

Always use [Azure AD security groups](#) as the assigned principal in an ACL entry. Resist the opportunity to directly assign individual users or service principals. Using this structure will allow you to add and remove users or service principals without the need to reapply ACLs to an entire directory structure. Instead, you can just add or remove users and service principals from the appropriate Azure AD security group.

There are many different ways to set up groups. For example, imagine that you have a directory named **/LogData** which holds log data that is generated by your server. Azure Data Factory (ADF) ingests data into that folder. Specific users from the service engineering team will upload logs and manage other users of this folder, and various Databricks clusters will analyze logs from that folder.

To enable these activities, you could create a **LogsWriter** group and a **LogsReader** group. Then, you could assign permissions as follows:

- Add the **LogsWriter** group to the ACL of the **/LogData** directory with **rwx** permissions.
- Add the **LogsReader** group to the ACL of the **/LogData** directory with **r-x** permissions.
- Add the service principal object or Managed Service Identity (MSI) for ADF to the **LogsWriters** group.
- Add users in the service engineering team to the **LogsWriter** group.
- Add the service principal object or MSI for Databricks to the **LogsReader** group.

If a user in the service engineering team leaves the company, you could just remove them from the **LogsWriter** group. If you did not add that user to a group, but instead, you added a dedicated ACL entry for that user, you would have to remove that ACL entry from the **/LogData** directory. You would also have to remove the entry from all subdirectories and files in the entire directory hierarchy of the **/LogData** directory.

To create a group and add members, see [Create a basic group and add members using Azure Active Directory](#).

## How are Azure RBAC and ACL permissions evaluated?

To learn how the system evaluates Azure RBAC and ACLs together to make authorization decisions for storage account resources, see [How permissions are evaluated](#).

## What are the limits for Azure role assignments and ACL entries?

The following table provides a summary view of the limits to consider while using Azure RBAC to manage "coarse-grained" permissions (permissions that apply to storage accounts or containers) and using ACLs to manage "fine-grained" permissions (permissions that apply to files and directories). Use security groups for ACL assignments. By using groups, you're less likely to exceed the maximum number of role assignments per subscription and the maximum number of ACL entries per file or directory.

MECHANISM	SCOPE	LIMITS	SUPPORTED LEVEL OF PERMISSION
Azure RBAC	Storage accounts, containers. Cross resource Azure role assignments at subscription or resource group level.	2000 Azure role assignments in a subscription	Azure roles (built-in or custom)

MECHANISM	SCOPE	LIMITS	SUPPORTED LEVEL OF PERMISSION
ACL	Directory, file	32 ACL entries (effectively 28 ACL entries) per file and per directory. Access and default ACLs each have their own 32 ACL entry limit.	ACL permission

### Does Data Lake Storage Gen2 support inheritance of Azure RBAC?

Azure role assignments do inherit. Assignments flow from subscription, resource group, and storage account resources down to the container resource.

### Does Data Lake Storage Gen2 support inheritance of ACLs?

Default ACLs can be used to set ACLs for new child subdirectories and files created under the parent directory. To update ACLs for existing child items, you will need to add, update, or remove ACLs recursively for the desired directory hierarchy. For guidance, see the [How to set ACLs](#) section of this article.

### Which permissions are required to recursively delete a directory and its contents?

- The caller has 'super-user' permissions,

Or

- The parent directory must have Write + Execute permissions.
- The directory to be deleted, and every directory within it, requires Read + Write + Execute permissions.

#### NOTE

You do not need Write permissions to delete files in directories. Also, the root directory "/" can never be deleted.

### Who is the owner of a file or directory?

The creator of a file or directory becomes the owner. In the case of the root directory, this is the identity of the user who created the container.

### Which group is set as the owning group of a file or directory at creation?

The owning group is copied from the owning group of the parent directory under which the new file or directory is created.

### I am the owning user of a file but I don't have the RWX permissions I need. What do I do?

The owning user can change the permissions of the file to give themselves any RWX permissions they need.

### Why do I sometimes see GUIDs in ACLs?

A GUID is shown if the entry represents a user and that user doesn't exist in Azure AD anymore. Usually this happens when the user has left the company or if their account has been deleted in Azure AD. Additionally, service principals and security groups do not have a User Principal Name (UPN) to identify them and so they are represented by their OID attribute (a guid).

### How do I set ACLs correctly for a service principal?

When you define ACLs for service principals, it's important to use the Object ID (OID) of the *service principal* for the app registration that you created. It's important to note that registered apps have a separate service principal in the specific Azure AD tenant. Registered apps have an OID that's visible in the Azure portal, but the *service principal* has another (different) OID.

To get the OID for the service principal that corresponds to an app registration, you can use the `az ad sp show`

command. Specify the Application ID as the parameter. Here's an example on obtaining the OID for the service principal that corresponds to an app registration with App ID = 18218b12-1895-43e9-ad80-6e8fc1ea88ce. Run the following command in the Azure CLI:

```
az ad sp show --id 18218b12-1895-43e9-ad80-6e8fc1ea88ce --query objectId
```

OID will be displayed.

When you have the correct OID for the service principal, go to the Storage Explorer **Manage Access** page to add the OID and assign appropriate permissions for the OID. Make sure you select **Save**.

### Can I set the ACL of a container?

No. A container does not have an ACL. However, you can set the ACL of the container's root directory. Every container has a root directory, and it shares the same name as the container. For example, if the container is named `my-container`, then the root directory is named `my-container/`.

The Azure Storage REST API does contain an operation named [Set Container ACL](#), but that operation cannot be used to set the ACL of a container or the root directory of a container. Instead, that operation is used to indicate whether blobs in a container [may be accessed publicly](#).

### Where can I learn more about POSIX access control model?

- [POSIX Access Control Lists on Linux](#)
- [HDFS permission guide](#)
- [POSIX FAQ](#)
- [POSIX 1003.1 2008](#)
- [POSIX 1003.1 2013](#)
- [POSIX 1003.1 2016](#)
- [POSIX ACL on Ubuntu](#)
- [ACL using access control lists on Linux](#)

## See also

- [Access control model in Azure Data Lake Storage Gen2](#)

# Multi-protocol access on Azure Data Lake Storage

8/22/2022 • 2 minutes to read • [Edit Online](#)

Blob APIs now work with accounts that have a hierarchical namespace. This unlocks the ecosystem of tools, applications, and services, as well as several Blob storage features to accounts that have a hierarchical namespace.

Until recently, you might have had to maintain separate storage solutions for object storage and analytics storage. That's because Azure Data Lake Storage Gen2 had limited ecosystem support. It also had limited access to Blob service features such as diagnostic logging. A fragmented storage solution is hard to maintain because you have to move data between accounts to accomplish various scenarios. You no longer have to do that.

With multi-protocol access on Data Lake Storage, you can work with your data by using the ecosystem of tools, applications, and services. This also includes third-party tools and applications. You can point them to accounts that have a hierarchical namespace without having to modify them. These applications work *as is* even if they call Blob APIs, because Blob APIs can now operate on data in accounts that have a hierarchical namespace.

Blob storage features such as [diagnostic logging](#), [access tiers](#), and [Blob storage lifecycle management policies](#) now work with accounts that have a hierarchical namespace. Therefore, you can enable hierarchical namespaces on your blob Storage accounts without losing access to these important features.

## NOTE

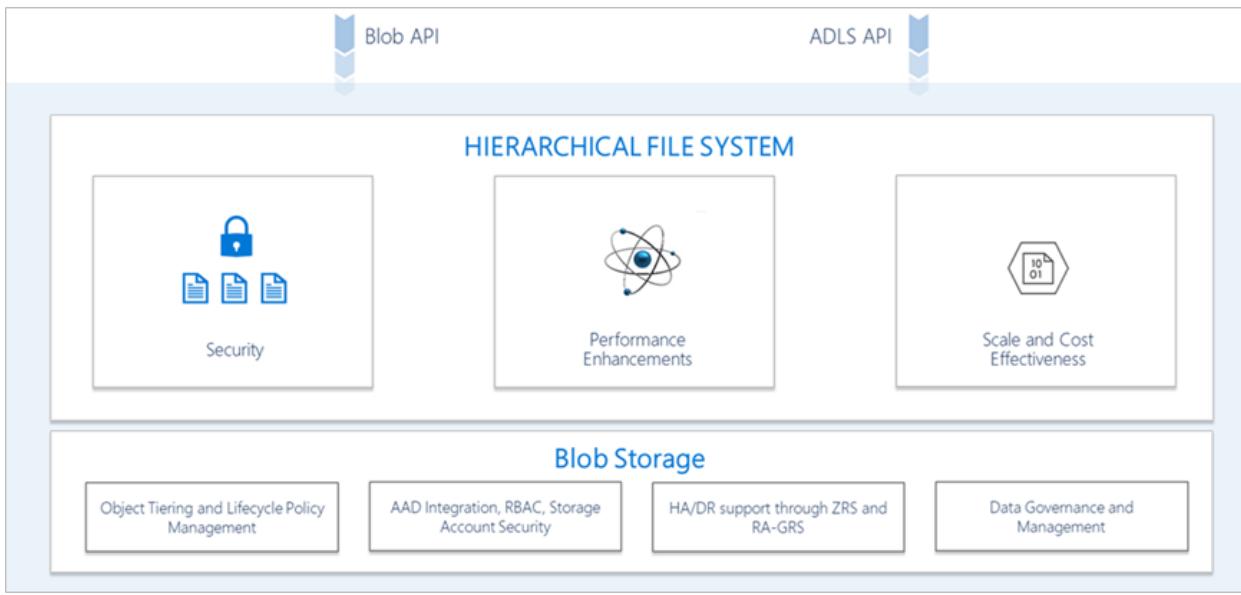
Multi-protocol access on Data Lake Storage is generally available and is available in all regions. Some Azure services or Blob storage features enabled by multi-protocol access remain in preview. These articles summarize the current support for Blob storage features and Azure service integrations.

[Blob Storage feature support in Azure Storage accounts](#)

[Azure services that support Azure Data Lake Storage Gen2](#)

## How multi-protocol access on data lake storage works

Blob APIs and Data Lake Storage Gen2 APIs can operate on the same data in storage accounts that have a hierarchical namespace. Data Lake Storage Gen2 routes Blob APIs through the hierarchical namespace so that you can get the benefits of first class directory operations and POSIX-compliant access control lists (ACLs).



Existing tools and applications that use the Blob API gain these benefits automatically. Developers won't have to modify them. Data Lake Storage Gen2 consistently applies directory and file-level ACLs regardless of the protocol that tools and applications use to access the data.

## See also

- [Blob Storage feature support in Azure Storage accounts](#)
- [Azure services that support Azure Data Lake Storage Gen2](#)
- [Open source platforms that support Azure Data Lake Storage Gen2](#)
- [Known issues with Azure Data Lake Storage Gen2](#)

# Blob Storage feature support in Azure Storage accounts

8/22/2022 • 4 minutes to read • [Edit Online](#)

Feature support is impacted by the type of account that you create and the settings that enable on that account. You can use the tables in this article to assess feature support based on these factors. The items that appear in these tables will change over time as support continues to expand.

## How to use these tables

Each table uses the following icons to indicate support level:

ICON	DESCRIPTION
✓	Fully supported
□	Supported at the preview level
□	Not <i>yet</i> supported

This table describes the impact of **enabling** the capability and not the specific use of that capability. For example, if you enable the Network File System (NFS) 3.0 protocol but never use the NFS 3.0 protocol to upload a blob, a check mark in the **NFS 3.0 enabled** column indicates that feature support is not negatively impacted by merely enabling NFS 3.0 support.

Even though a feature is not be negatively impacted, it might not be compatible when used with a specific capability. For example, enabling NFS 3.0 has no impact on Azure Active Directory (Azure AD) authorization. However, you can't use Azure AD to authorize an NFS 3.0 request. See any of these articles for information about known limitations:

- [Known issues: Hierarchical namespace capability](#)
- [Known issues: Network File System \(NFS\) 3.0 protocol](#)
- [Known issues: SSH File Transfer Protocol \(SFTP\)](#)

## Standard general-purpose v2 accounts

The following table describes whether a feature is supported in a standard general-purpose v2 account when you enable a hierarchical namespace (HNS), NFS 3.0 protocol, or SFTP.

### IMPORTANT

This table describes the impact of **enabling** HNS, NFS, or SFTP and not the specific use of those capabilities.

STORAGE FEATURE	DEFAULT	HNS	NFS	SFTP
Access tier - archive	✓	✓	✓	✓

Storage Feature	Default	HNS	NFS	SFTP
Access tier - cool	✓	✓	✓	✓
Access tier - hot	✓	✓	✓	✓
Anonymous public access	✓	✓	✓	✓
Azure Active Directory security	✓	✓	✓ <sup>1</sup>	✓ <sup>1</sup>
Blob inventory	✓	□	□	□
Blob index tags	✓	□	□	□
Blob snapshots	✓	□	□	□
Blob Storage APIs	✓	✓	✓	✓
Blob Storage Azure CLI commands	✓	✓	✓	✓
Blob Storage events	✓	✓	□	✓
Blob Storage PowerShell commands	✓	✓	✓	✓
Blob versioning	✓	□	□	□
Blobfuse	✓	✓	✓	✓
Change feed	✓	□	□	□
Custom domains	✓	□	□	□
Customer-managed account failover	✓	□	□	□
Customer-managed keys (encryption)	✓	✓	✓	✓
Customer-provided keys (encryption)	✓	□	□	□
Data redundancy options	✓	✓	✓ <sup>2</sup>	✓
Encryption scopes	✓	□	□	□
Immutable storage	✓	□	□	□

Storage feature	Default	HNS	NFS	SFTP
Last access time tracking for lifecycle management	✓	✓	□	✓
Lifecycle management policies (delete blob)	✓	✓	✓	✓
Lifecycle management policies (tiering)	✓	✓	✓	✓
Logging in Azure Monitor	✓	✓	□	□
Metrics in Azure Monitor	✓	✓	✓	✓
Object replication for block blobs	✓	□	□	□
Point-in-time restore for block blobs	✓	□	□	□
Soft delete for blobs	✓	✓	✓	✓
Soft delete for containers	✓	✓	✓	✓
Static websites	✓	✓	□	✓
Storage Analytics logs (classic)	✓	✓	□	✓
Storage Analytics metrics (classic)	✓	✓	✓	✓

<sup>1</sup> Requests that clients make by using NFS 3.0 or SFTP can't be authorized by using Azure Active Directory (AD) security.

<sup>2</sup> Only locally redundant storage (LRS) and zone-redundant storage (ZRS) are supported.

## Premium block blob accounts

The following table describes whether a feature is supported in a premium block blob account when you enable a hierarchical namespace (HNS), NFS 3.0 protocol, or SFTP.

### IMPORTANT

This table describes the impact of enabling HNS, NFS, or SFTP and not the specific use of those capabilities.

Storage feature	Default	HNS	NFS	SFTP
Access tier - archive	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Access tier - cool	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Access tier - hot	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Anonymous public access	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Azure Active Directory security	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <sup>1</sup>	<input checked="" type="checkbox"/> <sup>1</sup>
Blob inventory	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Blob index tags	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Blob snapshots	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Blob Storage APIs	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Blob Storage Azure CLI commands	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Blob Storage events	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Blob Storage PowerShell commands	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Blob versioning	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Blobfuse	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Change feed	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Custom domains	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Customer-managed account failover	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Customer-managed keys (encryption)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Customer-provided keys (encryption)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Data redundancy options	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <sup>2</sup>	<input checked="" type="checkbox"/>
Encryption scopes	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Immutable storage	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Storage feature	Default	HNS	NFS	SFTP
Last access time tracking for lifecycle management	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Lifecycle management policies (delete blob)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Lifecycle management policies (tiering)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Logging in Azure Monitor	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Metrics in Azure Monitor	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Object replication for block blobs	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Point-in-time restore for block blobs	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Soft delete for blobs	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Soft delete for containers	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Static websites	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Storage Analytics logs (classic)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Storage Analytics metrics (classic)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

<sup>1</sup> Requests that clients make by using NFS 3.0 or SFTP can't be authorized by using Azure Active Directory (AD) security.

<sup>2</sup> Only locally redundant storage (LRS) and zone-redundant storage (ZRS) are supported.

## See also

- [Known issues with Azure Data Lake Storage Gen2](#)
- [Known issues with Network File System \(NFS\) 3.0 protocol support in Azure Blob Storage](#)
- [Known issues with SSH File Transfer Protocol \(SFTP\) support in Azure Blob Storage \(preview\)](#)

# Azure services that support Azure Data Lake Storage Gen2

8/22/2022 • 2 minutes to read • [Edit Online](#)

You can use Azure services to ingest data, perform analytics, and create visual representations. This article provides a list of supported Azure services, discloses their level of support, and provides you with links to articles that help you to use these services with Azure Data Lake Storage Gen2.

## Supported Azure services

This table lists the Azure services that you can use with Azure Data Lake Storage Gen2. The items that appear in these tables will change over time as support continues to expand.

### NOTE

Support level refers only to how the service is supported with Data Lake Storage Gen 2.

AZURE SERVICE	SUPPORT LEVEL	AZURE AD	SHARED KEY	RELATED ARTICLES
Azure Data Factory	Generally available	Yes	Yes	<ul style="list-style-type: none"><li>• <a href="#">Load data into Azure Data Lake Storage Gen2 with Azure Data Factory</a></li></ul>
Azure Databricks	Generally available	Yes	Yes	<ul style="list-style-type: none"><li>• <a href="#">Use with Azure Databricks</a></li><li>• <a href="#">Tutorial: Extract, transform, and load data by using Azure Databricks</a></li><li>• <a href="#">Tutorial: Access Data Lake Storage Gen2 data with Azure Databricks using Spark</a></li></ul>

Azure Service	Support Level	Azure AD	Shared Key	Related Articles
Azure Event Hub	Generally available	No	Yes	<ul style="list-style-type: none"> <li><a href="#">Capture events through Azure Event Hubs in Azure Blob Storage or Azure Data Lake Storage</a></li> </ul>
Azure Event Grid	Generally available	Yes	Yes	<ul style="list-style-type: none"> <li><a href="#">Tutorial: Implement the data lake capture pattern to update a Databricks Delta table</a></li> </ul>
Azure Logic Apps	Generally available	No	Yes	<ul style="list-style-type: none"> <li><a href="#">Overview - What is Azure Logic Apps?</a></li> </ul>
Azure Machine Learning	Generally available	Yes	Yes	<ul style="list-style-type: none"> <li><a href="#">Access data in Azure storage services</a></li> </ul>
Azure Stream Analytics	Generally available	Yes	Yes	<ul style="list-style-type: none"> <li><a href="#">Quickstart: Create a Stream Analytics job by using the Azure portal</a></li> <li><a href="#">Egress to Azure Data Lake Gen2</a></li> </ul>
Data Box	Generally available	No	Yes	<ul style="list-style-type: none"> <li><a href="#">Use Azure Data Box to migrate data from an on-premises HDFS store to Azure Storage</a></li> </ul>

Azure Service	Support Level	Azure AD	Shared Key	Related Articles
HDInsight	Generally available	Yes	Yes	<ul style="list-style-type: none"> <li>• <a href="#">Azure Storage overview in HDInsight</a></li> <li>• <a href="#">Use Azure storage with Azure HDInsight clusters</a></li> <li>• <a href="#">Use Azure Data Lake Storage Gen2 with Azure HDInsight clusters</a></li> <li>• <a href="#">Using the HDFS CLI with Data Lake Storage Gen2</a></li> <li>• <a href="#">Tutorial: Extract, transform, and load data by using Apache Hive on Azure HDInsight</a></li> </ul>
IoT Hub	Generally available	Yes	Yes	<ul style="list-style-type: none"> <li>• <a href="#">Use IoT Hub message routing to send device-to-cloud messages to different endpoints</a></li> </ul>
Power BI	Generally available	Yes	Yes	<ul style="list-style-type: none"> <li>• <a href="#">Analyze data in Data Lake Storage Gen2 using Power BI</a></li> </ul>
Azure Synapse Analytics (formerly SQL Data Warehouse)	Generally available	Yes	Yes	<ul style="list-style-type: none"> <li>• <a href="#">Analyze data in a storage account</a></li> </ul>
SQL Server Integration Services (SSIS)	Generally available	Yes	Yes	<ul style="list-style-type: none"> <li>• <a href="#">Azure Storage connection manager</a></li> </ul>

AZURE SERVICE	SUPPORT LEVEL	AZURE AD	SHARED KEY	RELATED ARTICLES
Azure Data Explorer	Generally available	Yes	Yes	<ul style="list-style-type: none"> <li>• <a href="#">Query data in Azure Data Lake using Azure Data Explorer</a></li> </ul>
Azure Cognitive Search	Generally available	Yes	Yes	<ul style="list-style-type: none"> <li>• <a href="#">Index and search Azure Data Lake Storage Gen2 documents</a></li> </ul>
Azure SQL Managed Instance	Preview	No	Yes	<ul style="list-style-type: none"> <li>• <a href="#">Data virtualization with Azure SQL Managed Instance (preview)</a></li> </ul>
Azure Content Delivery Network	Not yet supported	Not applicable	Not applicable	<ul style="list-style-type: none"> <li>• <a href="#">Index and search Azure Data Lake Storage Gen2 documents (preview)</a></li> </ul>
Azure SQL Database	Not yet supported	Not applicable	Not applicable	<ul style="list-style-type: none"> <li>• <a href="#">What is Azure SQL Database?</a></li> </ul>

#### TIP

To see how services organized into categories such as ingest, download, process, and visualize, see [Ingest, process, and analyze](#).

## See also

- [Known issues with Azure Data Lake Storage Gen2](#)
- [Blob Storage feature support in Azure Storage accounts](#)
- [Open source platforms that support Azure Data Lake Storage Gen2](#)
- [Multi-protocol access on Azure Data Lake Storage](#)
- [Best practices for using Azure Data Lake Storage Gen2](#)

# Open source platforms that support Azure Data Lake Storage Gen2

8/22/2022 • 2 minutes to read • [Edit Online](#)

This article lists the open source platforms that support Data Lake Storage Gen2.

## Supported open source platforms

This table lists the open source platforms that support Data Lake Storage Gen2.

### NOTE

Only the versions that appear in this table are supported.

PLATFORM	SUPPORTED VERSION(S)	MORE INFORMATION
HDInsight	3.6+	<a href="#">What are the Apache Hadoop components and versions available with HDInsight?</a>
Hadoop	3.2+	<a href="#">Apache Hadoop releases archive</a>
Cloudera	6.1+	<a href="#">Cloudera Enterprise 6.x release notes</a>
Azure Databricks	5.1+	<a href="#">Databricks Runtime versions</a>
Hortonworks	3.1.x++	<a href="#">Configuring cloud data access</a>

## See also

- [Known issues with Azure Data Lake Storage Gen2](#)
- [Blob Storage feature support in Azure Storage accounts](#)
- [Azure services that support Azure Data Lake Storage Gen2](#)
- [Multi-protocol access on Azure Data Lake Storage](#)

# Monitoring Azure Blob Storage

8/22/2022 • 21 minutes to read • [Edit Online](#)

When you have critical applications and business processes that rely on Azure resources, you want to monitor those resources for their availability, performance, and operation. This article describes the monitoring data that's generated by Azure Blob Storage and how you can use the features of Azure Monitor to analyze alerts on this data.

## Monitor overview

The [Overview](#) page in the Azure portal for each Blob storage resource includes a brief view of the resource usage, such as requests and hourly billing. This information is useful, but only a small amount of the monitoring data is available. Some of this data is collected automatically and is available for analysis as soon as you create the resource. You can enable additional types of data collection with some configuration.

## What is Azure Monitor?

Azure Blob Storage creates monitoring data by using [Azure Monitor](#), which is a full stack monitoring service in Azure. Azure Monitor provides a complete set of features to monitor your Azure resources and resources in other clouds and on-premises.

Start with the article [Monitoring Azure resources with Azure Monitor](#) which describes the following:

- What is Azure Monitor?
- Costs associated with monitoring
- Monitoring data collected in Azure
- Configuring data collection
- Standard tools in Azure for analyzing and alerting on monitoring data

The following sections build on this article by describing the specific data gathered from Azure Storage. Examples show how to configure data collection and analyze this data with Azure tools.

## Monitoring data

Azure Blob Storage collects the same kinds of monitoring data as other Azure resources, which are described in [Monitoring data from Azure resources](#).

See [Azure Blob Storage monitoring data reference](#) for detailed information on the metrics and logs metrics created by Azure Blob Storage.

Metrics and logs in Azure Monitor support only Azure Resource Manager storage accounts. Azure Monitor doesn't support classic storage accounts. If you want to use metrics or logs on a classic storage account, you need to migrate to an Azure Resource Manager storage account. For more information, see [Migrate to Azure Resource Manager](#).

You can continue using classic metrics and logs if you want to. In fact, classic metrics and logs are available in parallel with metrics and logs in Azure Monitor. The support remains in place until Azure Storage ends the service on legacy metrics and logs.

## Collection and routing

Platform metrics and the Activity log are collected automatically, but can be routed to other locations by using a diagnostic setting.

To collect resource logs, you must create a diagnostic setting. When you create the setting, choose **blob** as the type of storage that you want to enable logs for. Then, specify one of the following categories of operations for which you want to collect logs.

CATEGORY	DESCRIPTION
StorageRead	Read operations on objects.
StorageWrite	Write operations on objects.
StorageDelete	Delete operations on objects.

**NOTE**

Data Lake Storage Gen2 doesn't appear as a storage type. That's because Data Lake Storage Gen2 is a set of capabilities available to Blob storage.

## Creating a diagnostic setting

This section shows you how to create a diagnostic setting by using the Azure portal, PowerShell, and the Azure CLI. This section provides steps specific to Azure Storage. For general guidance about how to create a diagnostic setting, see [Create diagnostic setting to collect platform logs and metrics in Azure](#).

**TIP**

You can also create a diagnostic setting by using an Azure Resource manager template or by using a policy definition. A policy definition can ensure that a diagnostic setting is created for every account that is created or updated.

This section doesn't describe templates or policy definitions.

- To view an Azure Resource Manager template that creates a diagnostic setting, see [Diagnostic setting for Azure Storage](#).
- To learn how to create a diagnostic setting by using a policy definition, see [Azure Policy built-in definitions for Azure Storage](#).

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

1. Sign in to the Azure portal.
2. Navigate to your storage account.
3. In the **Monitoring** section, click **Diagnostic settings**.

The screenshot shows the Azure portal interface with the title bar "Microsoft Azure". The left sidebar has sections for Home, Insights, Alerts, Metrics, Workbooks, and a highlighted "Diagnostic settings" option. The main content area shows a search bar and filter controls for Subscription (contoso), Resource group (contoso-resource-group), Resource type (Storage accounts), and Resource (contoso). Below these filters, a message says "Select any of the resources to view diagnostic settings." A table lists five resources: contoso, blob, queue, table, and file, all categorized as Storage accounts under the contoso-resource-group and marked as "Disabled" for diagnostics status. The "Diagnostic settings" link in the sidebar is highlighted with a red box.

4. Choose **blob** as the type of storage that you want to enable logs for.
5. Click **Add diagnostic setting**.

The **Diagnostic settings** page appears.

## Diagnostic setting



 Save  Discard  Delete  Feedback

A diagnostic setting specifies a list of categories of platform logs and/or metrics that you want to collect from a resource, and one or more destinations that you would stream them to. Normal usage charges for the destination will occur. [Learn more about the different log categories and contents of those logs](#)

Diagnostic setting name \*

### Logs

#### Categories

- StorageRead
- StorageWrite
- StorageDelete

#### Metrics

- Transaction

### Destination details

- Send to Log Analytics workspace
- Archive to a storage account
- Stream to an event hub
- Send to partner solution

6. In the **Name** field of the page, enter a name for this Resource log setting. Then, select which operations you want logged (read, write, and delete operations), and where you want the logs to be sent.

#### Archive logs to a storage account

If you choose to archive your logs to a storage account, you'll pay for the volume of logs that are sent to the storage account. For specific pricing, see the [Platform Logs](#) section of the [Azure Monitor pricing](#) page. You can't send logs to the same storage account that you are monitoring with this setting. This would lead to recursive logs in which a log entry describes the writing of another log entry. You must create an account or use another existing account to store log information.

1. Select the **Archive to a storage account** checkbox, and then select the **Configure** button.
2. In the **Storage account** drop-down list, select the storage account that you want to archive your logs to, and then select the **Save** button.

#### IMPORTANT

You can't set a retention policy. However, you can manage the retention policy of a log container by defining a lifecycle management policy. To learn how, see [Optimize costs by automating Azure Blob Storage access tiers](#).

#### NOTE

Before you choose a storage account as the export destination, see [Archive Azure resource logs](#) to understand prerequisites on the storage account.

#### Stream logs to Azure Event Hubs

If you choose to stream your logs to an event hub, you'll pay for the volume of logs that are sent to the event hub. For specific pricing, see the [Platform Logs](#) section of the [Azure Monitor pricing](#) page. You'll need access to an existing event hub, or you'll need to create one before you complete this step.

1. Select the **Stream to an event hub** checkbox, and then select the **Configure** button.
2. In the **Select an event hub** pane, choose the namespace, name, and policy name of the event hub that you want to stream your logs to.
3. Select the **Save** button.

#### Send logs to Azure Log Analytics

1. Select the **Send to Log Analytics** checkbox, select a log analytics workspace, and then select the **Save** button. You'll need access to an existing log analytics workspace, or you'll need to create one before you complete this step.

#### IMPORTANT

You can't set a retention policy. However, you can manage the data retention period of Log Analytics at the workspace level or even specify different retention settings by data type. To learn how, see [Change the data retention period](#).

#### Send to a partner solution

You can also send platform metrics and logs to certain Azure Monitor partners. You must first install a partner integration into your subscription. Configuration options will vary by partner. Check the [Azure Monitor partner integrations documentation](#) for details.

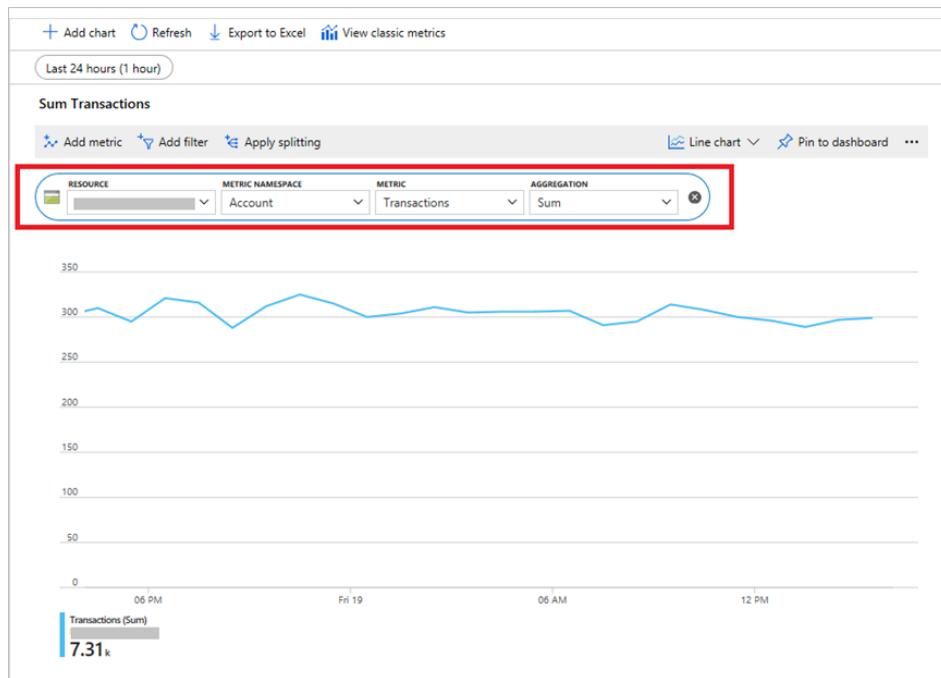
## Analyzing metrics

For a list of all Azure Monitor support metrics, which includes Azure Blob Storage, see [Azure Monitor supported metrics](#).

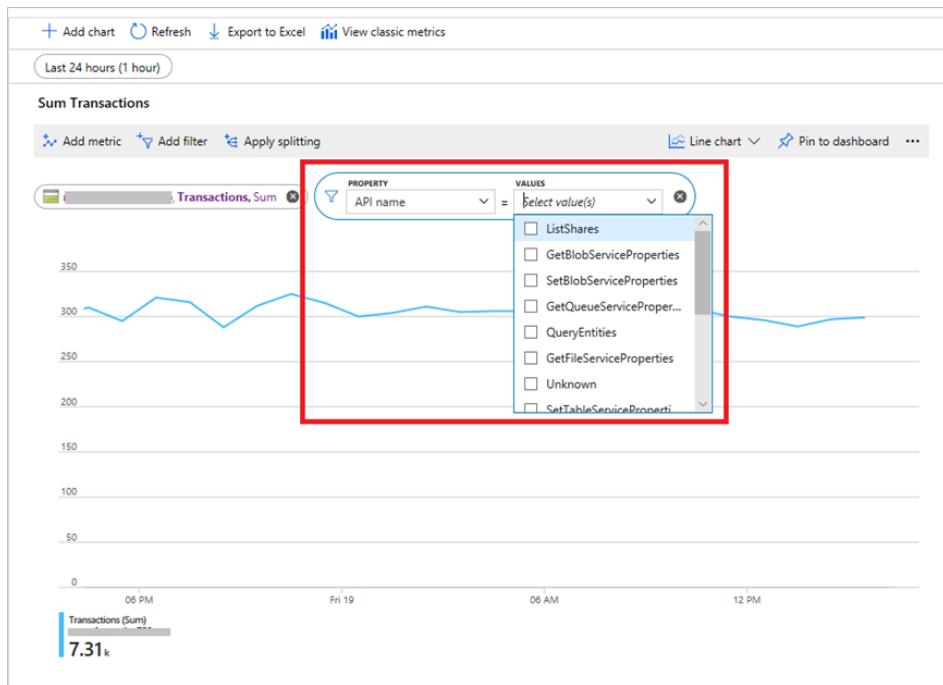
- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

You can analyze metrics for Azure Storage with metrics from other Azure services by using Metrics Explorer. Open Metrics Explorer by choosing **Metrics** from the **Azure Monitor** menu. For details on using this tool, see [Getting started with Azure Metrics Explorer](#).

This example shows how to view **Transactions** at the account level.



For metrics that support dimensions, you can filter the metric with the desired dimension value. This example shows how to view **Transactions** at the account level on a specific operation by selecting values for the **API Name** dimension.



For a complete list of the dimensions that Azure Storage supports, see [Metrics dimensions](#).

Metrics for Azure Blob Storage are in these namespaces:

- Microsoft.Storage/storageAccounts
- Microsoft.Storage/storageAccounts/blobServices

## Analyze metrics by using code

Azure Monitor provides the [.NET SDK](#) to read metric definition and values. The [sample code](#) shows how to use the SDK with different parameters. You need to use `0.18.0-preview` or a later version for storage metrics.

In these examples, replace the `<resource-ID>` placeholder with the resource ID of the entire storage account or the Blob storage service. You can find these resource IDs on the **Endpoints** pages of your storage account in the Azure portal.

Replace the `<subscription-ID>` variable with the ID of your subscription. For guidance on how to obtain values for `<tenant-ID>`, `<application-ID>`, and `<AccessKey>`, see [Use the portal to create an Azure AD application and service principal that can access resources](#).

### List the account-level metric definition

The following example shows how to list a metric definition at the account level:

```
public static async Task ListStorageMetricDefinition()
{
 var resourceId = "<resource-ID>";
 var subscriptionId = "<subscription-ID>";
 var tenantId = "<tenant-ID>";
 var applicationId = "<application-ID>";
 var accessKey = "<AccessKey>";

 MonitorManagementClient readOnlyClient = AuthenticateWithReadOnlyClient(tenantId, applicationId,
accessKey, subscriptionId).Result;
 IEnumerable<MetricDefinition> metricDefinitions = await
readOnlyClient.MetricDefinitions.ListAsync(resourceUri: resourceId, cancellationToken: new
CancellationToken());

 foreach (var metricDefinition in metricDefinitions)
 {
 // Enumerate metric definition:
 // Id
 // ResourceId
 // Name
 // Unit
 // MetricAvailabilities
 // PrimaryAggregationType
 // Dimensions
 // IsDimensionRequired
 }
}
```

### Reading account-level metric values

The following example shows how to read `UsedCapacity` data at the account level:

```

public static async Task ReadStorageMetricValue()
{
 var resourceId = "<resource-ID>";
 var subscriptionId = "<subscription-ID>";
 var tenantId = "<tenant-ID>";
 var applicationId = "<application-ID>";
 var accessKey = "<AccessKey>";

 MonitorClient readOnlyClient = AuthenticateWithReadOnlyClient(tenantId, applicationId, accessKey,
 subscriptionId).Result;

 Microsoft.Azure.Management.Monitor.Models.Response Response;

 string startDate = DateTime.Now.AddHours(-3).ToUniversalTime().ToString("o");
 string endDate = DateTime.Now.ToUniversalTime().ToString("o");
 string timeSpan = startDate + "/" + endDate;

 Response = await readOnlyClient.Metrics.ListAsync(
 resourceUri: resourceId,
 timespan: timeSpan,
 interval: System.TimeSpan.FromHours(1),
 metricnames: "UsedCapacity",

 aggregation: "Average",
 resultType: ResultType.Data,
 cancellationToken: CancellationToken.None);

 foreach (var metric in Response.Value)
 {
 // Enumerate metric value
 // Id
 // Name
 // Type
 // Unit
 // Timeseries
 // - Data
 // - Metadatavalues
 }
}

```

#### **Reading multidimensional metric values**

For multidimensional metrics, you need to define metadata filters if you want to read metric data on specific dimension values.

The following example shows how to read metric data on the metric supporting multidimension:

```

public static async Task ReadStorageMetricValueTest()
{
 // Resource ID for blob storage
 var resourceId =
"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccou
nts/{storageAccountName}/blobServices/default";
 var subscriptionId = "<subscription-ID>";
 // How to identify Tenant ID, Application ID and Access Key:
 https://azure.microsoft.com/documentation/articles/resource-group-create-service-principal-portal/
 var tenantId = "<tenant-ID>";
 var applicationId = "<application-ID>";
 var accessKey = "<AccessKey>";

 MonitorManagementClient readOnlyClient = AuthenticateWithReadOnlyClient(tenantId, applicationId,
accessKey, subscriptionId).Result;

 Microsoft.Azure.Management.Monitor.Models.Response Response;

 string startDate = DateTime.Now.AddHours(-3).ToUniversalTime().ToString("o");
 string endDate = DateTime.Now.ToUniversalTime().ToString("o");
 string timeSpan = startDate + "/" + endDate;
 // It's applicable to define meta data filter when a metric support dimension
 // More conditions can be added with the 'or' and 'and' operators, example: BlobType eq 'BlockBlob'
 or BlobType eq 'PageBlob'
 ODataQuery<MetadataValue> odataFilterMetrics = new ODataQuery<MetadataValue>(
 string.Format("BlobType eq '{0}'", "BlockBlob"));

 Response = readOnlyClient.Metrics.List(
 resourceUri: resourceId,
 timespan: timeSpan,
 interval: System.TimeSpan.FromHours(1),
 metricnames: "BlobCapacity",
 odataQuery: odataFilterMetrics,
 aggregation: "Average",
 resultType: ResultType.Data);

 foreach (var metric in Response.Value)
 {
 //Enumerate metric value
 // Id
 // Name
 // Type
 // Unit
 // Timeseries
 // - Data
 // - Metadatavalues
 }
}

```

## Analyzing logs

You can access resource logs either as a blob in a storage account, as event data, or through Log Analytics queries.

For a detailed reference of the fields that appear in these logs, see [Azure Blob Storage monitoring data reference](#).

Log entries are created only if there are requests made against the service endpoint. For example, if a storage account has activity in its blob endpoint but not in its table or queue endpoints, only logs that pertain to the blob service are created. Azure Storage logs contain detailed information about successful and failed requests to a storage service. This information can be used to monitor individual requests and to diagnose issues with a storage service. Requests are logged on a best-effort basis.

### Log authenticated requests

The following types of authenticated requests are logged:

- Successful requests
- Failed requests, including timeout, throttling, network, authorization, and other errors
- Requests that use a shared access signature (SAS) or OAuth, including failed and successful requests
- Requests to analytics data (classic log data in the **\$logs** container and class metric data in the **\$metric** tables)

Requests made by the Blob storage service itself, such as log creation or deletion, aren't logged. For a full list of the logged data, see [Storage logged operations and status messages](#) and [Storage log format](#).

### Log anonymous requests

The following types of anonymous requests are logged:

- Successful requests
- Server errors

- Timeout errors for both client and server
- Failed GET requests with the error code 304 (Not Modified)

All other failed anonymous requests aren't logged. For a full list of the logged data, see [Storage logged operations and status messages](#) and [Storage log format](#).

#### Accessing logs in a storage account

Logs appear as blobs stored to a container in the target storage account. Data is collected and stored inside a single blob as a line-delimited JSON payload. The name of the blob follows this naming convention:

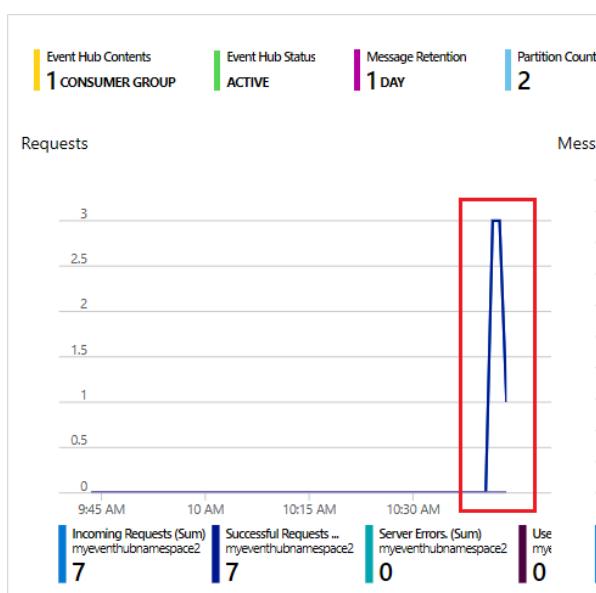
```
https://<destination-storage-account>.blob.core.windows.net/insights-logs-<storage-operation>/resourceId=/subscriptions/<subscription-ID>/resourceGroups/<resource-group-name>/providers/Microsoft.Storage/storageAccounts/<source-storage-account>/blobServices/default/y=<year>/m=<month>/d=<day>/h=<hour>/m=<minute>/PT1H.json
```

Here's an example:

```
https://mylogstorageaccount.blob.core.windows.net/insights-logs-storagewrite/resourceId=/subscriptions/208841be-a4v3-4234-9450-08b90c09f4/resourceGroups/myresourcegroup/providers/Microsoft.Storage/storageAccounts/mystorageaccount/blobServices/default/y=2019/m=07/d=30/
```

#### Accessing logs in an event hub

Logs sent to an event hub aren't stored as a file, but you can verify that the event hub received the log information. In the Azure portal, go to your event hub and verify that the **incoming messages** count is greater than zero.



You can access and read log data that's sent to your event hub by using security information and event management and monitoring tools. For more information, see [What can I do with the monitoring data being sent to my event hub?](#).

#### Accessing logs in a Log Analytics workspace

You can access logs sent to a Log Analytics workspace by using Azure Monitor log queries.

For more information, see [Get started with Log Analytics in Azure Monitor](#).

Data is stored in the **StorageBlobLog** table. Logs for Data Lake Storage Gen2 do not appear in a dedicated table. That's because Data Lake Storage Gen2 is not a service. It's a set of capabilities that you can enable in your storage account. If you've enabled those capabilities, logs will continue to appear in the **StorageBlobLog** table.

#### Sample Kusto queries

Here are some queries that you can enter in the **Log search** bar to help you monitor your Blob storage. These queries work with the [new language](#).

#### IMPORTANT

When you select **Logs** from the storage account resource group menu, Log Analytics is opened with the query scope set to the current resource group. This means that log queries will only include data from that resource group. If you want to run a query that includes data from other resources or data from other Azure services, select **Logs** from the **Azure Monitor** menu. See [Log query scope and time range in Azure Monitor Log Analytics](#) for details.

Use these queries to help you monitor your Azure Storage accounts:

- To list the 10 most common errors over the last three days.

```
StorageBlobLogs
| where TimeGenerated > ago(3d) and StatusText !contains "Success"
| summarize count() by StatusText
| top 10 by count_desc
```

- To list the top 10 operations that caused the most errors over the last three days.

```
StorageBlobLogs
| where TimeGenerated > ago(3d) and StatusText !contains "Success"
| summarize count() by OperationName
| top 10 by count_desc
```

- To list the top 10 operations with the longest end-to-end latency over the last three days.

```
StorageBlobLogs
| where TimeGenerated > ago(3d)
| top 10 by DurationMs desc
| project TimeGenerated, OperationName, DurationMs, ServerLatencyMs, ClientLatencyMs = DurationMs -
ServerLatencyMs
```

- To list all operations that caused server-side throttling errors over the last three days.

```
StorageBlobLogs
| where TimeGenerated > ago(3d) and StatusText contains "ServerBusy"
| project TimeGenerated, OperationName, StatusCode, StatusText
```

- To list all requests with anonymous access over the last three days.

```
StorageBlobLogs
| where TimeGenerated > ago(3d) and AuthenticationType == "Anonymous"
| project TimeGenerated, OperationName, AuthenticationType, Uri
```

- To create a pie chart of operations used over the last three days.

```
StorageBlobLogs
| where TimeGenerated > ago(3d)
| summarize count() by OperationName
| sort by count_desc
| render piechart
```

## Feature support

Support for this feature might be impacted by enabling Data Lake Storage Gen2, Network File System (NFS) 3.0 protocol, or the SSH File Transfer Protocol (SFTP).

If you've enabled any of these capabilities, see [Blob Storage feature support in Azure Storage accounts](#) to assess support for this feature.

## FAQ

### Does Azure Storage support metrics for Managed Disks or Unmanaged Disks?

No. Azure Compute supports the metrics on disks. For more information, see [Per disk metrics for Managed and Unmanaged Disks](#).

## Next steps

Get started with any of these guides.

GUIDE	DESCRIPTION
<a href="#">Gather metrics from your Azure Blob Storage containers</a>	Create charts that show metrics (Contains step-by-step guidance).
<a href="#">Monitor, diagnose, and troubleshoot your Azure Storage</a>	Troubleshoot storage account issues (contains step-by-step guidance).
<a href="#">Monitor storage with Azure Monitor Storage insights</a>	A unified view of storage performance, capacity, and availability
<a href="#">Best practices for monitoring Azure Blob Storage</a>	Guidance for common monitoring and troubleshooting scenarios.

GUIDE	DESCRIPTION
<a href="#">Getting started with Azure Metrics Explorer</a>	A tour of Metrics Explorer.
<a href="#">Overview of Log Analytics in Azure Monitor</a>	A tour of Log Analytics.
<a href="#">Azure Monitor Metrics overview</a>	The basics of metrics and metric dimensions
<a href="#">Azure Monitor Logs overview</a>	The basics of logs and how to collect and analyze them
<a href="#">Transition to metrics in Azure Monitor</a>	Move from Storage Analytics metrics to metrics in Azure Monitor.
<a href="#">Azure Blob Storage monitoring data reference</a>	A reference of the logs and metrics created by Azure Blob Storage
<a href="#">Troubleshoot performance issues</a>	Common performance issues and guidance about how to troubleshoot them.
<a href="#">Troubleshoot availability issues</a>	Common availability issues and guidance about how to troubleshoot them.
<a href="#">Troubleshoot client application errors</a>	Common issues with connecting clients and how to troubleshoot them.

# Transition to metrics in Azure Monitor

8/22/2022 • 4 minutes to read • [Edit Online](#)

On August 31, 2023 Storage Analytics metrics, also referred to as *classic metrics* will be retired. For more information, see the [official announcement](#). If you use classic metrics, make sure to transition to metrics in Azure Monitor prior to that date. This article helps you make the transition.

## Steps to complete the transition

To transition to metrics in Azure Monitor, we recommend the following approach.

1. Learn about some of the [key differences](#) between classic metrics and metrics in Azure Monitor.
2. Compile a list of classic metrics that you currently use.
3. Identify [which metrics in Azure Monitor](#) provide the same data as the metrics you currently use.
4. Create [charts](#) or [dashboards](#) to view metric data.

### NOTE

Metrics in Azure Monitor are enabled by default, so there is nothing you need to do to begin capturing metrics. You must however, create charts or dashboards to view those metrics.

5. If you've created alert rules that are based on classic storage metrics, then [create alert rules](#) that are based on metrics in Azure Monitor.
6. After you're able to see all of your metrics in Azure Monitor, you can turn off classic logging.

## Classic metrics vs. metrics in Azure Monitor

This section describes a few key differences between these two metrics platforms.

The main difference is in how metrics are managed. Classic metrics are managed by Azure Storage whereas metrics in Azure Monitor are managed by Azure Monitor. With classic metrics, Azure Storage collects metric values, aggregates them, and then stores them in tables that are located in the storage account. With metrics in Azure Monitor, Azure Storage sends metric data to the Azure Monitor back end. Azure Monitor provides a unified monitoring experience that includes data from the Azure portal as well as data that is ingested.

Classic metrics are sent and stored in an Azure storage account. Azure Monitor metrics can be sent to multiple locations. A storage account can be one of those locations, but it is not required.

As far as metrics support, classic metrics provide **capacity** metrics only for Azure Blob storage. Metrics in Azure Monitor provide capacity metrics for Blob, Table, File, Queue, and premium storage. Classic metrics provide **transaction** metrics on Blob, Table, Azure File, and Queue storage. Metrics in Azure Monitor add premium storage to that list.

If the activity in your account does not trigger a metric, classic metrics will show a value of zero (0) for that metric. Metrics in Azure Monitor will omit the data entirely, which leads to cleaner reports. For example, with classic metrics, if no server timeout errors are reported, then the `serverTimeoutError` value in the metrics table is set to 0. Azure Monitor doesn't return any data when you query the value of metric `Transactions` with dimension `ResponseType` equal to `ServerTimeoutError`.

To learn more about metrics in Azure Monitor, see [Metrics in Azure Monitor](#).

## Map classic metrics to metrics in Azure Monitor

Use these tables to identify which metrics in Azure Monitor provide the same data as the metrics you currently use.

### Capacity metrics

CLASSIC METRIC	METRIC IN AZURE MONITOR
Capacity	BlobCapacity with the dimension BlobType equal to BlockBlob or PageBlob
ObjectCount	BlobCount with the dimension BlobType equal to BlockBlob or PageBlob
ContainerCount	ContainerCount

#### NOTE

There are also several new capacity metrics that weren't available as classic metrics. To view the complete list, see [Metrics](#).

### Transaction metrics

CLASSIC METRIC	METRIC IN AZURE MONITOR
AnonymousAuthorizationError	Transactions with the dimension ResponseType equal to AuthorizationError and dimension Authentication equal to Anonymous
AnonymousClientOtherError	Transactions with the dimension ResponseType equal to ClientOtherError and dimension Authentication equal to Anonymous
AnonymousClientTimeoutError	Transactions with the dimension ResponseType equal to ClientTimeoutError and dimension Authentication equal to Anonymous
AnonymousNetworkError	Transactions with the dimension ResponseType equal to NetworkError and dimension Authentication equal to Anonymous
AnonymousServerOtherError	Transactions with the dimension ResponseType equal to ServerOtherError and dimension Authentication equal to Anonymous
AnonymousServerTimeoutError	Transactions with the dimension ResponseType equal to ServerTimeoutError and dimension Authentication equal to Anonymous

CLASSIC METRIC	METRIC IN AZURE MONITOR
AnonymousSuccess	Transactions with the dimension <code>ResponseType</code> equal to <code>Success</code> and dimension <code>Authentication</code> equal to <code>Anonymous</code>
AnonymousThrottlingError	Transactions with the dimension <code>ResponseType</code> equal to <code>ClientThrottlingError</code> or <code>ServerBusyError</code> and dimension <code>Authentication</code> equal to <code>Anonymous</code>
AuthorizationError	Transactions with the dimension <code>ResponseType</code> equal to <code>AuthorizationError</code>
Availability	Availability
AverageE2ELatency	SuccessE2ELatency
AverageServerLatency	SuccessServerLatency
ClientOtherError	Transactions with the dimension <code>ResponseType</code> equal to <code>ClientOtherError</code>
ClientTimeoutError	Transactions with the dimension <code>ResponseType</code> equal to <code>ClientTimeoutError</code>
NetworkError	Transactions with the dimension <code>ResponseType</code> equal to <code>NetworkError</code>
PercentAuthorizationError	Transactions with the dimension <code>ResponseType</code> equal to <code>AuthorizationError</code>
PercentClientOtherError	Transactions with the dimension <code>ResponseType</code> equal to <code>ClientOtherError</code>
PercentNetworkError	Transactions with the dimension <code>ResponseType</code> equal to <code>NetworkError</code>
PercentServerOtherError	Transactions with the dimension <code>ResponseType</code> equal to <code>ServerOtherError</code>
PercentSuccess	Transactions with the dimension <code>ResponseType</code> equal to <code>Success</code>
PercentThrottlingError	Transactions with the dimension <code>ResponseType</code> equal to <code>ClientThrottlingError</code> or <code>ServerBusyError</code>
PercentTimeoutError	Transactions with the dimension <code>ResponseType</code> equal to <code>ServerError</code> or <code>ResponseType</code> equal to <code>ClientTimeoutError</code>
SASAuthorizationError	Transactions with the dimension <code>ResponseType</code> equal to <code>AuthorizationError</code> and dimension <code>Authentication</code> equal to <code>SAS</code>

CLASSIC METRIC	METRIC IN AZURE MONITOR
SASClientOtherError	Transactions with the dimension <code>ResponseType</code> equal to <code>ClientOtherError</code> and dimension <code>Authentication</code> equal to <code>SAS</code>
SASClientTimeoutError	Transactions with the dimension <code>ResponseType</code> equal to <code>ClientTimeoutError</code> and dimension <code>Authentication</code> equal to <code>SAS</code>
SASNetworkError	Transactions with the dimension <code>ResponseType</code> equal to <code>NetworkError</code> and dimension <code>Authentication</code> equal to <code>SAS</code>
SASServerOtherError	Transactions with the dimension <code>ResponseType</code> equal to <code>ServerOtherError</code> and dimension <code>Authentication</code> equal to <code>SAS</code>
SASServerTimeoutError	Transactions with the dimension <code>ResponseType</code> equal to <code>ServerTimeoutError</code> and dimension <code>Authentication</code> equal to <code>SAS</code>
SASSuccess	Transactions with the dimension <code>ResponseType</code> equal to <code>Success</code> and dimension <code>Authentication</code> equal to <code>SAS</code>
SASThrottlingError	Transactions with the dimension <code>ResponseType</code> equal to <code>ClientThrottlingError</code> or <code>ServerBusyError</code> and dimension <code>Authentication</code> equal to <code>SAS</code>
ServerOtherError	Transactions with the dimension <code>ResponseType</code> equal to <code>ServerOtherError</code>
ServerTimeoutError	Transactions with the dimension <code>ResponseType</code> equal to <code>ServerTimeoutError</code>
Success	Transactions with the dimension <code>ResponseType</code> equal to <code>Success</code>
ThrottlingError	Transactions with the dimension <code>ResponseType</code> equal to <code>ClientThrottlingError</code> or <code>ServerBusyError</code>
TotalBillableRequests	Transactions
TotalEgress	Egress
TotalIngress	Ingress
TotalRequests	Transactions

## Next steps

- [Azure Monitor](#)

# Storage Analytics

8/22/2022 • 2 minutes to read • [Edit Online](#)

Azure Storage Analytics performs logging and provides metrics data for a storage account. You can use this data to trace requests, analyze usage trends, and diagnose issues with your storage account.

To use Storage Analytics, you must enable it individually for each service you want to monitor. You can enable it from the [Azure portal](#). For details, see [Monitor a storage account in the Azure portal](#). You can also enable Storage Analytics programmatically via the REST API or the client library. Use the [Set Blob Service Properties](#), [Set Queue Service Properties](#), [Set Table Service Properties](#), and [Set File Service Properties](#) operations to enable Storage Analytics for each service.

The aggregated data is stored in a well-known blob (for logging) and in well-known tables (for metrics), which may be accessed using the Blob service and Table service APIs.

Storage Analytics has a 20 TB limit on the amount of stored data that is independent of the total limit for your storage account. For more information about storage account limits, see [Scalability and performance targets for standard storage accounts](#).

For an in-depth guide on using Storage Analytics and other tools to identify, diagnose, and troubleshoot Azure Storage-related issues, see [Monitor, diagnose, and troubleshoot Microsoft Azure Storage](#).

## Billing for Storage Analytics

All metrics data is written by the services of a storage account. As a result, each write operation performed by Storage Analytics is billable. Additionally, the amount of storage used by metrics data is also billable.

The following actions performed by Storage Analytics are billable:

- Requests to create blobs for logging.
- Requests to create table entities for metrics.

If you have configured a data retention policy, you can reduce the spending by deleting old logging and metrics data. For more information about retention policies, see [Setting a Storage Analytics Data Retention Policy](#).

### Understanding billable requests

Every request made to an account's storage service is either billable or non-billable. Storage Analytics logs each individual request made to a service, including a status message that indicates how the request was handled. Similarly, Storage Analytics stores metrics for both a service and the API operations of that service, including the percentages and count of certain status messages. Together, these features can help you analyze your billable requests, make improvements on your application, and diagnose issues with requests to your services. For more information about billing, see [Understanding Azure Storage Billing - Bandwidth, Transactions, and Capacity](#).

When looking at Storage Analytics data, you can use the tables in the [Storage Analytics Logged Operations and Status Messages](#) topic to determine what requests are billable. Then you can compare your logs and metrics data to the status messages to see if you were charged for a particular request. You can also use the tables in the previous topic to investigate availability for a storage service or individual API operation.

## Next steps

- [Monitor a storage account in the Azure portal](#)
- [Storage Analytics Metrics](#)

- Storage Analytics Logging

# Azure Storage Analytics metrics (classic)

8/22/2022 • 5 minutes to read • [Edit Online](#)

On August 31, 2023 Storage Analytics metrics, also referred to as *classic metrics* will be retired. For more information, see the [official announcement](#). If you use classic metrics, make sure to transition to metrics in Azure Monitor prior to that date. This article helps you make the transition.

Azure Storage uses the Storage Analytics solution to store metrics that include aggregated transaction statistics and capacity data about requests to a storage service. Transactions are reported at the API operation level and at the storage service level. Capacity is reported at the storage service level. Metrics data can be used to:

- Analyze storage service usage.
- Diagnose issues with requests made against the storage service.
- Improve the performance of applications that use a service.

Storage Analytics metrics are enabled by default for new storage accounts. You can configure metrics in the [Azure portal](#), by using PowerShell, or by using the Azure CLI. For step-by-step guidance, see [Enable and manage Azure Storage Analytic metrics \(classic\)](#). You can also enable Storage Analytics programmatically via the REST API or the client library. Use the Set Service Properties operations to enable Storage Analytics for each service.

## NOTE

Storage Analytics metrics are available for Azure Blob storage, Azure Queue storage, Azure Table storage, and Azure Files. Storage Analytics metrics are now classic metrics. We recommend that you use [storage metrics in Azure Monitor](#) instead of Storage Analytics metrics.

## Transaction metrics

A robust set of data is recorded at hourly or minute intervals for each storage service and requested API operation, which includes ingress and egress, availability, errors, and categorized request percentages. For a complete list of the transaction details, see [Storage Analytics metrics table schema](#).

Transaction data is recorded at the service level and the API operation level. At the service level, statistics that summarize all requested API operations are written to a table entity every hour, even if no requests were made to the service. At the API operation level, statistics are only written to an entity if the operation was requested within that hour.

For example, if you perform a **GetBlob** operation on your blob service, Storage Analytics Metrics logs the request and includes it in the aggregated data for the blob service and the **GetBlob** operation. If no **GetBlob** operation is requested during the hour, an entity isn't written to **\$MetricsTransactionsBlob** for that operation.

Transaction metrics are recorded for user requests and requests made by Storage Analytics itself. For example, requests by Storage Analytics to write logs and table entities are recorded.

## Capacity metrics

## NOTE

Currently, capacity metrics are available only for the blob service.

Capacity data is recorded daily for a storage account's blob service, and two table entities are written. One entity provides statistics for user data, and the other provides statistics about the `$logs` blob container used by Storage Analytics. The `$MetricsCapacityBlob` table includes the following statistics:

- **Capacity**: The amount of storage used by the storage account's blob service, in bytes.
- **ContainerCount**: The number of blob containers in the storage account's blob service.
- **ObjectCount**: The number of committed and uncommitted block or page blobs in the storage account's blob service.

For more information about capacity metrics, see [Storage Analytics metrics table schema](#).

## How metrics are stored

All metrics data for each of the storage services is stored in three tables reserved for that service. One table is for transaction information, one table is for minute transaction information, and another table is for capacity information. Transaction and minute transaction information consists of request and response data. Capacity information consists of storage usage data. Hour metrics, minute metrics, and capacity for a storage account's blob service is accessed in tables that are named as described in the following table.

METRICS LEVEL	TABLE NAMES	SUPPORTED FOR VERSIONS
Hourly metrics, primary location	- <code>\$MetricsTransactionsBlob</code> - <code>\$MetricsTransactionsTable</code> - <code>\$MetricsTransactionsQueue</code>	Versions prior to August 15, 2013, only. While these names are still supported, we recommend that you switch to using the tables that follow.
Hourly metrics, primary location	- <code>\$MetricsHourPrimaryTransactionsBlob</code> - <code>\$MetricsHourPrimaryTransactionsTable</code> - <code>\$MetricsHourPrimaryTransactionsQueue</code> - <code>\$MetricsHourPrimaryTransactionsFile</code>	All versions. Support for file service metrics is available only in version April 5, 2015, and later.
Minute metrics, primary location	- <code>\$MetricsMinutePrimaryTransactionsBlob</code> - <code>\$MetricsMinutePrimaryTransactionsTable</code> - <code>\$MetricsMinutePrimaryTransactionsQueue</code> - <code>\$MetricsMinutePrimaryTransactionsFile</code>	All versions. Support for file service metrics is available only in version April 5, 2015, and later.
Hourly metrics, secondary location	- <code>\$MetricsHourSecondaryTransactionsBlob</code> - <code>\$MetricsHourSecondaryTransactionsTable</code> - <code>\$MetricsHourSecondaryTransactionsQueue</code>	All versions. Read-access geo-redundant replication must be enabled.

METRICS LEVEL	TABLE NAMES	SUPPORTED FOR VERSIONS
Minute metrics, secondary location	- \$MetricsMinuteSecondaryTransactions Blob - \$MetricsMinuteSecondaryTransactions Table - \$MetricsMinuteSecondaryTransactions Queue	All versions. Read-access geo-redundant replication must be enabled.
Capacity (blob service only)	\$MetricsCapacityBlob	All versions.

These tables are automatically created when Storage Analytics is enabled for a storage service endpoint. They're accessed via the namespace of the storage account, for example,

`https://<accountname>.table.core.windows.net/Tables("$MetricsTransactionsBlob")`. The metrics tables don't appear in a listing operation and must be accessed directly via the table name.

## Metrics alerts

Consider setting up alerts in the [Azure portal](#) so you'll be automatically notified of important changes in the behavior of your storage services. For step-by-step guidance, see [Create metrics alerts](#).

If you use a Storage Explorer tool to download this metrics data in a delimited format, you can use Microsoft Excel to analyze the data. For a list of available Storage Explorer tools, see [Azure Storage client tools](#).

### IMPORTANT

There might be a delay between a storage event and when the corresponding hourly or minute metrics data is recorded. In the case of minute metrics, several minutes of data might be written at once. This issue can lead to transactions from earlier minutes being aggregated into the transaction for the current minute. When this issue happens, the alert service might not have all available metrics data for the configured alert interval, which might lead to alerts firing unexpectedly.

## Billing on storage metrics

Write requests to create table entities for metrics are charged at the standard rates applicable to all Azure Storage operations.

Read requests of metrics data by a client are also billable at standard rates.

The capacity used by the metrics tables is also billable. Use the following information to estimate the amount of capacity used for storing metrics data:

- If each hour a service utilizes every API in every service, approximately 148 KB of data is stored every hour in the metrics transaction tables if you enabled a service-level and API-level summary.
- If within each hour a service utilizes every API in the service, approximately 12 KB of data is stored every hour in the metrics transaction tables if you enabled only a service-level summary.
- The capacity table for blobs has two rows added each day provided you opted in for logs. This scenario implies that every day the size of this table increases by up to approximately 300 bytes.

## Next steps

- [Storage Analytics metrics table schema](#)
- [Storage Analytics logged operations and status messages](#)

- Storage Analytics logging

# Azure Storage analytics logging

8/22/2022 • 6 minutes to read • [Edit Online](#)

Storage Analytics logs detailed information about successful and failed requests to a storage service. This information can be used to monitor individual requests and to diagnose issues with a storage service. Requests are logged on a best-effort basis. This means that most requests will result in a log record, but the completeness and timeliness of Storage Analytics logs are not guaranteed.

## NOTE

We recommend that you use Azure Storage logs in Azure Monitor instead of Storage Analytics logs. To learn more, see any of the following articles:

- [Monitoring Azure Blob Storage](#)
- [Monitoring Azure Files](#)
- [Monitoring Azure Queue Storage](#)
- [Monitoring Azure Table storage](#)

Storage Analytics logging is not enabled by default for your storage account. You can enable it in the [Azure portal](#) or by using PowerShell, or Azure CLI. For step-by-step guidance, see [Enable and manage Azure Storage Analytics logs \(classic\)](#).

You can also enable Storage Analytics logs programmatically via the REST API or the client library. Use the [Get Blob Service Properties](#), [Get Queue Service Properties](#), and [Get Table Service Properties](#) operations to enable Storage Analytics for each service. To see an example that enables Storage Analytics logs by using .NET, see [Enable logs](#)

Log entries are created only if there are requests made against the service endpoint. For example, if a storage account has activity in its Blob endpoint but not in its Table or Queue endpoints, only logs pertaining to the Blob service will be created.

## NOTE

Storage Analytics logging is currently available only for the Blob, Queue, and Table services. Storage Analytics logging is also available for premium-performance [BlockBlobStorage](#) accounts. However, it isn't available for general-purpose v2 accounts with premium performance.

## Requests logged in logging

### Logging authenticated requests

The following types of authenticated requests are logged:

- Successful requests
- Failed requests, including timeout, throttling, network, authorization, and other errors
- Requests using a Shared Access Signature (SAS) or OAuth, including failed and successful requests
- Requests to analytics data

Requests made by Storage Analytics itself, such as log creation or deletion, are not logged. A full list of the logged data is documented in the [Storage Analytics Logged Operations and Status Messages](#) and [Storage Analytics Log Format](#) topics.

### Logging anonymous requests

The following types of anonymous requests are logged:

- Successful requests
- Server errors
- Timeout errors for both client and server
- Failed GET requests with error code 304 (Not Modified)

All other failed anonymous requests are not logged. A full list of the logged data is documented in the [Storage Analytics Logged Operations and Status Messages](#) and [Storage Analytics Log Format](#) topics.

## NOTE

Storage Analytics logs all internal calls to the data plane. Calls from the Azure Storage Resource Provider are also logged. To identify these requests, look for the query string `<sk=system-1>` in the request URL.

## How logs are stored

All logs are stored in block blobs in a container named `$logs`, which is automatically created when Storage Analytics is enabled for a storage account. The `$logs` container is located in the blob namespace of the storage account, for example: [http://<accountname>.blob.core.windows.net/\\$logs](http://<accountname>.blob.core.windows.net/$logs). This container cannot be deleted once Storage Analytics has been enabled, though its contents can be deleted. If you use your storage-browsing tool to navigate to the container directly, you will see all the blobs that contain your logging data.

**NOTE**

The `$logs` container is not displayed when a container listing operation is performed, such as the List Containers operation. It must be accessed directly. For example, you can use the List Blobs operation to access the blobs in the `$logs` container.

As requests are logged, Storage Analytics will upload intermediate results as blocks. Periodically, Storage Analytics will commit these blocks and make them available as a blob. It can take up to an hour for log data to appear in the blobs in the `$logs` container because the frequency at which the storage service flushes the log writers. Duplicate records may exist for logs created in the same hour. You can determine if a record is a duplicate by checking the `RequestId` and `Operation` number.

If you have a high volume of log data with multiple files for each hour, then you can use the blob metadata to determine what data the log contains by examining the blob metadata fields. This is also useful because there can sometimes be a delay while data is written to the log files: the blob metadata gives a more accurate indication of the blob content than the blob name.

Most storage browsing tools enable you to view the metadata of blobs; you can also read this information using PowerShell or programmatically. The following PowerShell snippet is an example of filtering the list of log blobs by name to specify a time, and by metadata to identify just those logs that contain `write` operations.

```
Get-AzStorageBlob -Container '$logs' |
Where-Object {
 $_.Name -match 'blob/2014/05/21/05' -and
 $_.ICloudBlob.Metadata.LogType -match 'write'
} |
ForEach-Object {
 "{0} {1} {2} {3}" -f $_.Name,
 $_.ICloudBlob.Metadata.StartTime,
 $_.ICloudBlob.Metadata.EndTime,
 $_.ICloudBlob.Metadata.LogType
}
```

For information about listing blobs programmatically, see [Enumerating Blob Resources](#) and [Setting and Retrieving Properties and Metadata for Blob Resources](#).

#### Log naming conventions

Each log will be written in the following format:

```
<service-name>/YYYY/MM/DD/hhmm/<counter>.log
```

The following table describes each attribute in the log name:

ATTRIBUTE	DESCRIPTION
<code>&lt;service-name&gt;</code>	The name of the storage service. For example: <code>blob</code> , <code>table</code> , or <code>queue</code>
<code>YYYY</code>	The four digit year for the log. For example: <code>2011</code>
<code>MM</code>	The two digit month for the log. For example: <code>07</code>
<code>DD</code>	The two digit day for the log. For example: <code>31</code>
<code>hh</code>	The two digit hour that indicates the starting hour for the logs, in 24 hour UTC format. For example: <code>18</code>
<code>mm</code>	The two digit number that indicates the starting minute for the logs. <b>Note:</b> This value is unsupported in the current version of Storage Analytics, and its value will always be <code>00</code> .
<code>&lt;counter&gt;</code>	A zero-based counter with six digits that indicates the number of log blobs generated for the storage service in an hour time period. This counter starts at <code>000000</code> . For example: <code>000001</code>

The following is a complete sample log name that combines the above examples:

```
blob/2011/07/31/1800/000001.log
```

The following is a sample URI that can be used to access the above log:

```
https://<accountname>.blob.core.windows.net/$logs/blob/2011/07/31/1800/000001.log
```

When a storage request is logged, the resulting log name correlates to the hour when the requested operation completed. For example, if a GetBlob request was completed at 6:30PM on 7/31/2011, the log would be written with the following prefix: `blob/2011/07/31/1800/`

### Log metadata

All log blobs are stored with metadata that can be used to identify what logging data the blob contains. The following table describes each metadata attribute:

ATTRIBUTE	DESCRIPTION
<code>LogType</code>	Describes whether the log contains information pertaining to read, write, or delete operations. This value can include one type or a combination of all three, separated by commas.  Example 1: <code>write</code>  Example 2: <code>read,write</code>  Example 3: <code>read,write,delete</code>
<code>StartTime</code>	The earliest time of an entry in the log, in the form of <code>YYYY-MM-DDThh:mm:ssZ</code> . For example:  <code>2011-07-31T18:21:46Z</code>
<code>EndTime</code>	The latest time of an entry in the log, in the form of <code>YYYY-MM-DDThh:mm:ssZ</code> . For example:  <code>2011-07-31T18:22:09Z</code>
<code>LogVersion</code>	The version of the log format.

The following list displays complete sample metadata using the above examples:

- `LogType=write`
- `StartTime=2011-07-31T18:21:46Z`
- `EndTime=2011-07-31T18:22:09Z`
- `LogVersion=1.0`

### Log entries

The following sections show an example log entry for each supported Azure Storage service.

#### Example log entry for Blob Storage

```
2.0;2022-01-03T0:34:54.4617505Z;PutBlob;SASSuccess;201;7;7;sas;;logsamples/blob;https://logsamples.blob.core.windows.net/container1/1.txt?se=2022-02-02T20:34:54Z&sig=XXXXXX&sp=rw&sr=c&sv=2020-04-08&timeout=901;" /logsamples/container1/1.txt";xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx;0;71.197.193.44:53371;2019-12-12;654;13;37;0;13;"xxxxxxxxxxxxxxxxxxxxx=";"xxxxxxxxxxxxxxxxxxxxx=";""0x8D9CEF88004E296"" ;Monday, 03-Jan-22 20:34:54 GMT;"Microsoft Azure Storage Explorer, 1.20.1, win32, azcopy-node, 2.0.0, win32, AzCopy/10.11.0 Azure-Storage/0.13 (go1.15; Windows_NT) ";"xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxx";;;;;;
```

#### Example log entry for Blob Storage (Data Lake Storage Gen2 enabled)

```
2.0;2022-01-04T22:50:56.0000775Z;RenamePathFile;Success;201;49;49;authenticated;logsamples;logsamples/blob;"https://logsamples.dfs.core.windows.net/my-container/myfileorig.png?mode=legacy";"/logsamples/my-container/myfilerenamed.png";xxxxxxxx-xxxx-xxxx-xxxxxxxxxx;0;73.157.16.8;2020-04-08;591;0;224;0;0;;;;Friday, 11-Jun-21 17:58:15 GMT;"Microsoft Azure Storage Explorer, 1.19.1, win32 azsdk-js-storagedatalake/12.3.1 (NODE-VERSION v12.16.3; Windows_NT 10.0.22000) ";"xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxx";;;;;;
```

#### Example log entry for Queue Storage

```
2.0;2022-01-03T20:35:04.6097590Z;PeekMessages;Success;200;5;authenticated;logsamples;logsamples;queue;https://logsamples.queue.core.windows.net/queue1/messages;numofmessages=32&peekonly=true&timeout=30;" /logsamples/queue1";xxxxxxxx-xxxx-xxxx-xxxxxxxxxx;0;71.197.193.44:53385;2020-04-08;536;0;232;62;0;;;;;"Microsoft Azure Storage Explorer, 1.20.1, win32 azsdk-js-storagequeue/12.3.1 (NODE-VERSION v12.16.3; Windows_NT 10.0.22000) ";"xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxx";;;;;;
```

#### Example log entry for Table Storage

```
1.0;2022-01-03T20:35:13.0719766Z;CreateTable;Success;204;30;30;authenticated;logsamples;logsamples;table;https://logsamples.table.core.windows.net/Tables;" /logxxxx-xxxx-xxxx-xxxxxxxxxx;0;71.197.193.44:53389;2018-03-28;601;22;339;0;22;;;;;"Microsoft Azure Storage Explorer, 1.20.1, win32, Azure-Storage/2.12.16.3; Windows_NT 10.0.22000) ";"xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxx"
```

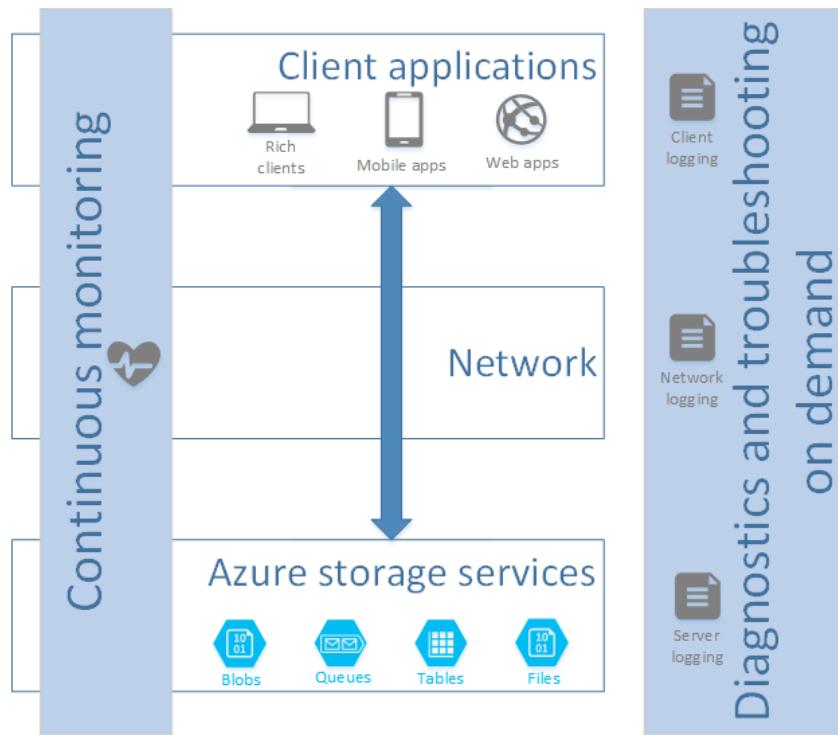
## Next steps

- [Enable and manage Azure Storage Analytics logs \(classic\)](#)
- [Storage Analytics Log Format](#)
- [Storage Analytics Logged Operations and Status Messages](#)
- [Storage Analytics Metrics \(classic\)](#)

# Monitor, diagnose, and troubleshoot Microsoft Azure Storage (classic)

8/22/2022 • 51 minutes to read • [Edit Online](#)

This guide shows you how to use features such as Azure Storage Analytics, client-side logging in the Azure Storage Client Library, and other third-party tools to identify, diagnose, and troubleshoot Azure Storage related issues.



This guide is intended to be read primarily by developers of online services that use Azure Storage Services and IT Pros responsible for managing such online services. The goals of this guide are:

- To help you maintain the health and performance of your Azure Storage accounts.
- To provide you with the necessary processes and tools to help you decide whether an issue or problem in an application relates to Azure Storage.
- To provide you with actionable guidance for resolving problems related to Azure Storage.

## NOTE

This article is based on using Storage Analytics metrics and logs as referred to as *Classic metrics and logs*. We recommend that you use Azure Storage metrics and logs in Azure Monitor instead of Storage Analytics logs. To learn more, see any of the following articles:

- [Monitoring Azure Blob Storage](#)
- [Monitoring Azure Files](#)
- [Monitoring Azure Queue Storage](#)
- [Monitoring Azure Table storage](#)

## Overview

Diagnosing and troubleshooting issues in a distributed application hosted in a cloud environment can be more complex than in traditional environments. Applications can be deployed in a PaaS or IaaS infrastructure, on-premises, on a mobile device, or in some combination of these environments. Typically, your application's network traffic may traverse public and private networks and your application may use multiple storage technologies such as Microsoft Azure Storage Tables, Blobs, Queues, or Files in addition to other data stores such as relational and document databases.

To manage such applications successfully you should monitor them proactively and understand how to diagnose and troubleshoot all aspects of them and their dependent technologies. As a user of Azure Storage services, you should continuously monitor the Storage services your application uses for any unexpected changes in behavior (such as slower than usual response times), and use logging to collect more detailed data and to analyze a problem in depth. The diagnostics information you obtain from both monitoring and logging will help you to determine the root cause of the issue your application encountered. Then you can troubleshoot the issue and determine the appropriate steps you can take to remediate it. Azure Storage is a core Azure service, and forms an important part of the majority of solutions that customers deploy to the Azure infrastructure. Azure Storage includes capabilities to simplify monitoring, diagnosing, and troubleshooting storage issues in your cloud-based applications.

### How this guide is organized

The section "[Monitoring your storage service](#)" describes how to monitor the health and performance of your Azure Storage services using Azure Storage Analytics Metrics (Storage Metrics).

The section "[Diagnosing storage issues](#)" describes how to diagnose issues using Azure Storage Analytics Logging (Storage Logging). It also describes how to enable client-side logging using the facilities in one of the client libraries such as the Storage Client Library for .NET or the Azure SDK for Java.

The section "[End-to-end tracing](#)" describes how you can correlate the information contained in various log files and metrics data.

The section "[Troubleshooting guidance](#)" provides troubleshooting guidance for some of the common storage-related issues you might encounter.

The "[Appendices](#)" include information about using other tools such as Wireshark and Netmon for analyzing network packet data, and Fiddler for analyzing HTTP/HTTPS messages.

## Monitoring your storage service

If you are familiar with Windows performance monitoring, you can think of Storage Metrics as being an Azure Storage equivalent of Windows Performance Monitor counters. In Storage Metrics, you will find a comprehensive set of metrics (counters in Windows Performance Monitor terminology) such as service availability, total number of requests to service, or percentage of successful requests to service. For a full list of the available metrics, see [Storage Analytics Metrics Table Schema](#). You can specify whether you want the storage service to collect and aggregate metrics every hour or every minute. For more information about how to enable metrics and monitor your storage accounts, see [Enabling storage metrics and viewing metrics data](#).

You can choose which hourly metrics you want to display in the [Azure portal](#) and configure rules that notify administrators by email whenever an hourly metric exceeds a particular threshold. For more information, see [Receive Alert Notifications](#).

We recommend you review [Azure Monitor for Storage](#) (preview). It is a feature of Azure Monitor that offers comprehensive monitoring of your Azure Storage accounts by delivering a unified view of your Azure Storage services performance, capacity, and availability. It does not require you to enable or configure anything, and you can immediately view these metrics from the pre-defined interactive charts and other visualizations included.

The storage service collects metrics using a best effort, but may not record every storage operation.

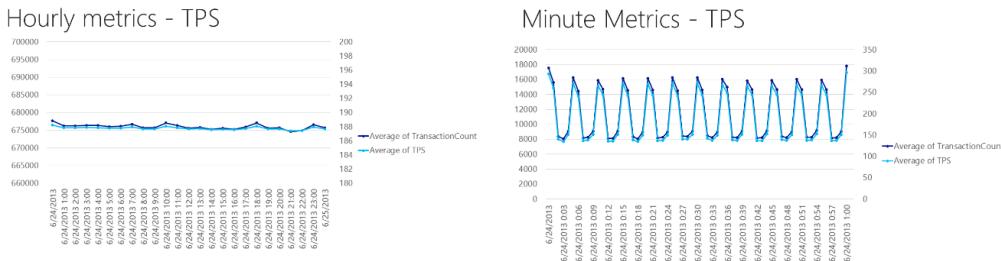
In the Azure portal, you can view metrics such as availability, total requests, and average latency numbers for a storage account. A notification rule has also been set up to alert an administrator if availability drops below a certain level. From viewing this data, one possible area for investigation is the table service success percentage being below 100% (for more information, see the section "[Metrics show low PercentSuccess or analytics log entries have operations with transaction status of ClientOtherErrors](#)").

You should continuously monitor your Azure applications to ensure they are healthy and performing as expected by:

- Establishing some baseline metrics for application that will enable you to compare current data and identify any significant changes in the behavior of Azure storage and your application. The values of your baseline metrics will, in many cases, be application specific and you should establish them when you are performance testing your application.
- Recording minute metrics and using them to monitor actively for unexpected errors and anomalies such as spikes in error counts or request rates.
- Recording hourly metrics and using them to monitor average values such as average error counts and request rates.

- Investigating potential issues using diagnostics tools as discussed later in the section "[Diagnosing storage issues](#)."

The charts in the following image illustrate how the averaging that occurs for hourly metrics can hide spikes in activity. The hourly metrics appear to show a steady rate of requests, while the minute metrics reveal the fluctuations that are really taking place.



The remainder of this section describes what metrics you should monitor and why.

### Monitoring service health

You can use the [Azure portal](#) to view the health of the Storage service (and other Azure services) in all the Azure regions around the world. Monitoring enables you to see immediately if an issue outside of your control is affecting the Storage service in the region you use for your application.

The [Azure portal](#) can also provide notifications of incidents that affect the various Azure services. Note: This information was previously available, along with historical data, on the [Azure Service Dashboard](#). For more information about Application Insights for Azure DevOps, see the appendix "[Appendix 5: Monitoring with Application Insights for Azure DevOps](#)."

### Monitoring capacity

Storage Metrics only stores capacity metrics for the blob service because blobs typically account for the largest proportion of stored data (at the time of writing, it is not possible to use Storage Metrics to monitor the capacity of your tables and queues). You can find this data in the **\$MetricsCapacityBlob** table if you have enabled monitoring for the Blob service. Storage Metrics records this data once per day, and you can use the value of the **RowKey** to determine whether the row contains an entity that relates to user data (value **data**) or analytics data (value **analytics**). Each stored entity contains information about the amount of storage used (**Capacity** measured in bytes) and the current number of containers (**ContainerCount**) and blobs (**ObjectCount**) in use in the storage account. For more information about the capacity metrics stored in the **\$MetricsCapacityBlob** table, see [Storage Analytics Metrics Table Schema](#).

#### NOTE

You should monitor these values for an early warning that you are approaching the capacity limits of your storage account. In the Azure portal, you can add alert rules to notify you if aggregate storage use exceeds or falls below thresholds that you specify.

For help estimating the size of various storage objects such as blobs, see the blog post [Understanding Azure Storage Billing – Bandwidth, Transactions, and Capacity](#).

### Monitoring availability

You should monitor the availability of the storage services in your storage account by monitoring the value in the **Availability** column in the hourly or minute metrics tables — **\$MetricsHourPrimaryTransactionsBlob**, **\$MetricsHourPrimaryTransactionsTable**, **\$MetricsHourPrimaryTransactionsQueue**, **\$MetricsMinutePrimaryTransactionsBlob**, **\$MetricsMinutePrimaryTransactionsTable**, **\$MetricsMinutePrimaryTransactionsQueue**, **\$MetricsCapacityBlob**. The **Availability** column contains a percentage value that indicates the availability of the service or the API operation represented by the row (the **RowKey** shows if the row contains metrics for the service as a whole or for a specific API operation).

Any value less than 100% indicates that some storage requests are failing. You can see why they are failing by examining the other columns in the metrics data that show the numbers of requests with different error types such as **ServerTimeoutError**. You should expect to see **Availability** fall temporarily below 100% for reasons such as transient server timeouts while the service moves partitions to better load-balance request; the retry logic in your client application should handle such intermittent conditions. The article [Storage Analytics Logged Operations and Status Messages](#) lists the transaction types that Storage Metrics includes in its **Availability** calculation.

In the [Azure portal](#), you can add alert rules to notify you if **Availability** for a service falls below a threshold that you specify.

The "Troubleshooting guidance" section of this guide describes some common storage service issues related to availability.

### Monitoring performance

To monitor the performance of the storage services, you can use the following metrics from the hourly and minute metrics tables.

- The values in the **AverageE2ELatency** and **AverageServerLatency** columns show the average time the storage service or API operation type is taking to process requests. **AverageE2ELatency** is a measure of end-to-end latency that includes the time taken to read the request and send the response in addition to the time taken to process the request (therefore includes network latency once the request reaches the storage service); **AverageServerLatency** is a measure of just the processing time and therefore excludes any network latency related to communicating with the client. See the section "[Metrics show high AverageE2ELatency and low AverageServerLatency](#)" later in this guide for a discussion of why there might be a significant difference between these two values.
- The values in the **TotalIngress** and **TotalEgress** columns show the total amount of data, in bytes, coming in to and going out of your storage service or through a specific API operation type.
- The values in the **TotalRequests** column show the total number of requests that the storage service or API operation is receiving. **TotalRequests** is the total number of requests that the storage service receives.

Typically, you will monitor for unexpected changes in any of these values as an indicator that you have an issue that requires investigation.

In the [Azure portal](#), you can add alert rules to notify you if any of the performance metrics for this service fall below or exceed a threshold that you specify.

The "Troubleshooting guidance" section of this guide describes some common storage service issues related to performance.

## Diagnosing storage issues

There are a number of ways that you might become aware of a problem or issue in your application, including:

- A major failure that causes the application to crash or to stop working.
- Significant changes from baseline values in the metrics you are monitoring as described in the previous section "[Monitoring your storage service](#)."
- Reports from users of your application that some particular operation didn't complete as expected or that some feature is not working.
- Errors generated within your application that appear in log files or through some other notification method.

Typically, issues related to Azure storage services fall into one of four broad categories:

- Your application has a performance issue, either reported by your users, or revealed by changes in the performance metrics.
- There is a problem with the Azure Storage infrastructure in one or more regions.
- Your application is encountering an error, either reported by your users, or revealed by an increase in one of the error count metrics you monitor.
- During development and test, you may be using the local storage emulator; you may encounter some issues that relate specifically to usage of the storage emulator.

The following sections outline the steps you should follow to diagnose and troubleshoot issues in each of these four categories. The section "[Troubleshooting guidance](#)" later in this guide provides more detail for some common issues you may encounter.

### Service health issues

Service health issues are typically outside of your control. The [Azure portal](#) provides information about any ongoing issues with Azure services including storage services. If you opted for Read-Access Geo-Redundant Storage when you created your storage account, then if your data becomes unavailable in the primary location, your application can switch temporarily to the read-only copy in the secondary location. To read from the secondary, your application must be able to switch between using the primary and secondary storage locations, and be able to work in a reduced functionality mode with read-only data. The Azure Storage Client libraries allow you to define a retry policy that can read from secondary storage in case a read from primary storage

fails. Your application also needs to be aware that the data in the secondary location is eventually consistent. For more information, see the blog post [Azure Storage Redundancy Options and Read Access Geo Redundant Storage](#).

### Performance issues

The performance of an application can be subjective, especially from a user perspective. Therefore, it is important to have baseline metrics available to help you identify where there might be a performance issue. Many factors might affect the performance of an Azure storage service from the client application perspective. These factors might operate in the storage service, in the client, or in the network infrastructure; therefore it is important to have a strategy for identifying the origin of the performance issue.

After you have identified the likely location of the cause of the performance issue from the metrics, you can then use the log files to find detailed information to diagnose and troubleshoot the problem further.

The section "[Troubleshooting guidance](#)" later in this guide provides more information about some common performance-related issues you may encounter.

### Diagnosing errors

Users of your application may notify you of errors reported by the client application. Storage Metrics also records counts of different error types from your storage services such as **NetworkError**, **ClientTimeoutError**, or **AuthorizationError**. While Storage Metrics only records counts of different error types, you can obtain more detail about individual requests by examining server-side, client-side, and network logs. Typically, the HTTP status code returned by the storage service will give an indication of why the request failed.

#### NOTE

Remember that you should expect to see some intermittent errors: for example, errors due to transient network conditions, or application errors.

The following resources are useful for understanding storage-related status and error codes:

- [Common REST API Error Codes](#)
- [Blob Service Error Codes](#)
- [Queue Service Error Codes](#)
- [Table Service Error Codes](#)
- [File Service Error Codes](#)

### Storage emulator issues

The Azure SDK includes a storage emulator you can run on a development workstation. This emulator simulates most of the behavior of the Azure storage services and is useful during development and test, enabling you to run applications that use Azure storage services without the need for an Azure subscription and an Azure storage account.

The "[Troubleshooting guidance](#)" section of this guide describes some common issues encountered using the storage emulator.

### Storage logging tools

Storage Logging provides server-side logging of storage requests in your Azure storage account. For more information about how to enable server-side logging and access the log data, see [Enabling Storage Logging and Accessing Log Data](#).

The Storage Client Library for .NET enables you to collect client-side log data that relates to storage operations performed by your application. For more information, see [Client-side Logging with the .NET Storage Client Library](#).

#### NOTE

In some circumstances (such as SAS authorization failures), a user may report an error for which you can find no request data in the server-side Storage logs. You can use the logging capabilities of the Storage Client Library to investigate if the cause of the issue is on the client or use network monitoring tools to investigate the network.

### Using network logging tools

You can capture the traffic between the client and server to provide detailed information about the data the client and server are exchanging and the underlying network conditions. Useful network logging tools include:

- [Fiddler](#) is a free web debugging proxy that enables you to examine the headers and payload data of HTTP and HTTPS request and response messages. For more information, see [Appendix 1: Using Fiddler to capture HTTP and HTTPS traffic](#).
- [Microsoft Network Monitor \(Netmon\)](#) and [Wireshark](#) are free network protocol analyzers that enable you to view detailed packet information for a wide range of network protocols. For more information about Wireshark, see ["Appendix 2: Using Wireshark to capture network traffic"](#).
- If you want to perform a basic connectivity test to check that your client machine can connect to the Azure storage service over the network, you cannot do this using the standard `ping` tool on the client. However, you can use the [tcping tool](#) to check connectivity.

In many cases, the log data from Storage Logging and the Storage Client Library will be sufficient to diagnose an issue, but in some scenarios, you may need the more detailed information that these network logging tools can provide. For example, using Fiddler to view HTTP and HTTPS messages enables you to view header and payload data sent to and from the storage services, which would enable you to examine how a client application retries storage operations. Protocol analyzers such as Wireshark operate at the packet level enabling you to view TCP data, which would enable you to troubleshoot lost packets and connectivity issues.

## End-to-end tracing

End-to-end tracing using a variety of log files is a useful technique for investigating potential issues. You can use the date/time information from your metrics data as an indication of where to start looking in the log files for the detailed information that will help you troubleshoot the issue.

### Correlating log data

When viewing logs from client applications, network traces, and server-side storage logging it is critical to be able to correlate requests across the different log files. The log files include a number of different fields that are useful as correlation identifiers. The client request ID is the most useful field to use to correlate entries in the different logs. However sometimes, it can be useful to use either the server request ID or timestamps. The following sections provide more details about these options.

### Client request ID

The Storage Client Library automatically generates a unique client request ID for every request.

- In the client-side log that the Storage Client Library creates, the client request ID appears in the **Client Request ID** field of every log entry relating to the request.
- In a network trace such as one captured by Fiddler, the client request ID is visible in request messages as the `x-ms-client-request-id` HTTP header value.
- In the server-side Storage Logging log, the client request ID appears in the Client request ID column.

#### NOTE

It is possible for multiple requests to share the same client request ID because the client can assign this value (although the Storage Client Library assigns a new value automatically). When the client retries, all attempts share the same client request ID. In the case of a batch sent from the client, the batch has a single client request ID.

### Server request ID

The storage service automatically generates server request IDs.

- In the server-side Storage Logging log, the server request ID appears the **Request ID header** column.
- In a network trace such as one captured by Fiddler, the server request ID appears in response messages as the `x-ms-request-id` HTTP header value.
- In the client-side log that the Storage Client Library creates, the server request ID appears in the **Operation Text** column for the log entry showing details of the server response.

#### NOTE

The storage service always assigns a unique server request ID to every request it receives, so every retry attempt from the client and every operation included in a batch has a unique server request ID.

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

The code sample below demonstrates how to use a custom client request ID.

```

var connectionString = Constants.connectionString;

BlobServiceClient blobServiceClient = new BlobServiceClient(connectionString);

BlobContainerClient blobContainerClient = blobServiceClient.GetBlobContainerClient("demcontainer");

BlobClient blobClient = blobContainerClient.GetBlobClient("testImage.jpg");

string clientRequestID = String.Format("{0} {1} {2} {3}", HOSTNAME, APPNAME, USERID,
Guid.NewGuid().ToString());

using (HttpPipeline.CreateClientRequestIdScope(clientRequestID))
{
 BlobDownloadInfo download = blobClient.Download();

 using (FileStream downloadFileStream = File.OpenWrite("C:\\\\testImage.jpg"))
 {
 download.Content.CopyTo(downloadFileStream);
 downloadFileStream.Close();
 }
}

```

## Timestamps

You can also use timestamps to locate related log entries, but be careful of any clock skew between the client and server that may exist. Search plus or minus 15 minutes for matching server-side entries based on the timestamp on the client. Remember that the blob metadata for the blobs containing metrics indicates the time range for the metrics stored in the blob. This time range is useful if you have many metrics blobs for the same minute or hour.

## Troubleshooting guidance

This section will help you with the diagnosis and troubleshooting of some of the common issues your application may encounter when using the Azure storage services. Use the list below to locate the information relevant to your specific issue.

### Troubleshooting Decision Tree

---

Does your issue relate to the performance of one of the storage services?

- Metrics show high AverageE2ELatency and low AverageServerLatency
  - Metrics show low AverageE2ELatency and low AverageServerLatency but the client is experiencing high latency
  - Metrics show high AverageServerLatency
  - You are experiencing unexpected delays in message delivery on a queue
- 

Does your issue relate to the availability of one of the storage services?

- Metrics show an increase in PercentThrottlingError
  - Metrics show an increase in PercentTimeoutError
  - Metrics show an increase in PercentNetworkError
- 

Is your client application receiving an HTTP 4XX (such as 404) response from a storage service?

- The client is receiving HTTP 403 (Forbidden) messages
  - The client is receiving HTTP 404 (Not found) messages
  - The client is receiving HTTP 409 (Conflict) messages
- 

Metrics show low PercentSuccess or analytics log entries have operations with transaction status of ClientOtherErrors

---

Capacity metrics show an unexpected increase in storage capacity usage

---

Your issue arises from using the storage emulator for development or test

---

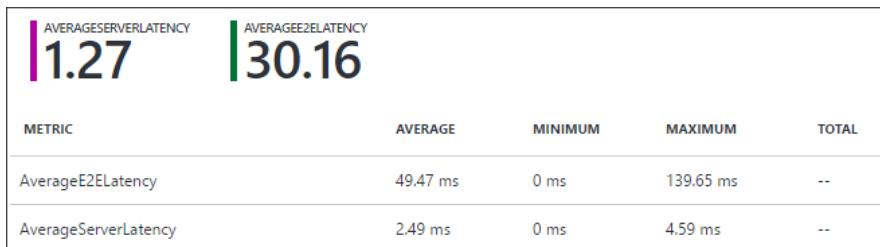
You are encountering problems installing the Azure SDK for .NET

---

You have a different issue with a storage service

### Metrics show high AverageE2ELatency and low AverageServerLatency

The illustration below from the [Azure portal](#) monitoring tool shows an example where the **AverageE2ELatency** is significantly higher than the **AverageServerLatency**.



The storage service only calculates the metric **AverageE2ELatency** for successful requests and, unlike **AverageServerLatency**, includes the time the client takes to send the data and receive acknowledgment from the storage service. Therefore, a difference between **AverageE2ELatency** and **AverageServerLatency** could be either due to the client application being slow to respond, or due to conditions on the network.

#### NOTE

You can also view **E2ELatency** and **ServerLatency** for individual storage operations in the Storage Logging log data.

#### Investigating client performance issues

Possible reasons for the client responding slowly include having a limited number of available connections or threads, or being low on resources such as CPU, memory or network bandwidth. You may be able to resolve the issue by modifying the client code to be more efficient (for example by using asynchronous calls to the storage service), or by using a larger Virtual Machine (with more cores and more memory).

For the table and queue services, the Nagle algorithm can also cause high **AverageE2ELatency** as compared to **AverageServerLatency**: for more information, see the post [Nagle's Algorithm is Not Friendly towards Small Requests](#). You can disable the Nagle algorithm in code by using the **ServicePointManager** class in the **System.Net** namespace. You should do this before you make any calls to the table or queue services in your application since this does not affect connections that are already open. The following example comes from the **Application\_Start** method in a worker role.

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

```
var connectionString = Constants.connectionString;
QueueServiceClient queueServiceClient = new QueueServiceClient(connectionString);
ServicePoint queueServicePoint = ServicePointManager.FindServicePoint(queueServiceClient.Uri);
queueServicePoint.UseNagleAlgorithm = false;
```

You should check the client-side logs to see how many requests your client application is submitting, and check for general .NET related performance bottlenecks in your client such as CPU, .NET garbage collection, network utilization, or memory. As a starting point for troubleshooting .NET client applications, see [Debugging, Tracing, and Profiling](#).

#### Investigating network latency issues

Typically, high end-to-end latency caused by the network is due to transient conditions. You can investigate both transient and persistent network issues such as dropped packets by using tools such as Wireshark.

For more information about using Wireshark to troubleshoot network issues, see "[Appendix 2: Using Wireshark to capture network traffic](#)."

### Metrics show low AverageE2ELatency and low AverageServerLatency but the client is experiencing high latency

In this scenario, the most likely cause is a delay in the storage requests reaching the storage service. You should investigate why requests from the client are not making it through to the blob service.

One possible reason for the client delaying sending requests is that there are a limited number of available

connections or threads.

Also check whether the client is performing multiple retries, and investigate the reason if it is. To determine whether the client is performing multiple retries, you can:

- Examine the Storage Analytics logs. If multiple retries are happening, you will see multiple operations with the same client request ID but with different server request IDs.
- Examine the client logs. Verbose logging will indicate that a retry has occurred.
- Debug your code, and check the properties of the **OperationContext** object associated with the request. If the operation has retried, the **RequestResults** property will include multiple unique server request IDs. You can also check the start and end times for each request. For more information, see the code sample in the section [Server request ID](#).

If there are no issues in the client, you should investigate potential network issues such as packet loss. You can use tools such as Wireshark to investigate network issues.

For more information about using Wireshark to troubleshoot network issues, see "[Appendix 2: Using Wireshark to capture network traffic](#)."

#### Metrics show high AverageServerLatency

In the case of high **AverageServerLatency** for blob download requests, you should use the Storage Logging logs to see if there are repeated requests for the same blob (or set of blobs). For blob upload requests, you should investigate what block size the client is using (for example, blocks less than 64 K in size can result in overheads unless the reads are also in less than 64 K chunks), and if multiple clients are uploading blocks to the same blob in parallel. You should also check the per-minute metrics for spikes in the number of requests that result in exceeding the per second scalability targets: also see "[Metrics show an increase in PercentTimeoutError](#)."

If you are seeing high **AverageServerLatency** for blob download requests when there are repeated requests to the same blob or set of blobs, then you should consider caching these blobs using Azure Cache or the Azure Content Delivery Network (CDN). For upload requests, you can improve the throughput by using a larger block size. For queries to tables, it is also possible to implement client-side caching on clients that perform the same query operations and where the data doesn't change frequently.

High **AverageServerLatency** values can also be a symptom of poorly designed tables or queries that result in scan operations or that follow the append/prepend anti-pattern. For more information, see "[Metrics show an increase in PercentThrottlingError](#)".

#### NOTE

You can find a comprehensive checklist performance checklist here: [Microsoft Azure Storage Performance and Scalability Checklist](#).

#### You are experiencing unexpected delays in message delivery on a queue

If you are experiencing a delay between the time an application adds a message to a queue and the time it becomes available to read from the queue, then you should take the following steps to diagnose the issue:

- Verify the application is successfully adding the messages to the queue. Check that the application is not retrying the **AddMessage** method several times before succeeding. The Storage Client Library logs will show any repeated retries of storage operations.
- Verify there is no clock skew between the worker role that adds the message to the queue and the worker role that reads the message from the queue that makes it appear as if there is a delay in processing.
- Check if the worker role that reads the messages from the queue is failing. If a queue client calls the **GetMessage** method but fails to respond with an acknowledgment, the message will remain invisible on the queue until the **visibilityTimeout** period expires. At this point, the message becomes available for processing again.
- Check if the queue length is growing over time. This can occur if you do not have sufficient workers available to process all of the messages that other workers are placing on the queue. Also check the metrics to see if delete requests are failing and the dequeue count on messages, which might indicate repeated failed attempts to delete the message.
- Examine the Storage Logging logs for any queue operations that have higher than expected **E2ELatency** and **ServerLatency** values over a longer period of time than usual.

#### Metrics show an increase in PercentThrottlingError

Throttling errors occur when you exceed the scalability targets of a storage service. The storage service throttles

to ensure that no single client or tenant can use the service at the expense of others. For more information, see [Scalability and performance targets for standard storage accounts](#) for details on scalability targets for storage accounts and performance targets for partitions within storage accounts.

If the **PercentThrottlingError** metric shows an increase in the percentage of requests that are failing with a throttling error, you need to investigate one of two scenarios:

- [Transient increase in PercentThrottlingError](#)
- [Permanent increase in PercentThrottlingError error](#)

An increase in **PercentThrottlingError** often occurs at the same time as an increase in the number of storage requests, or when you are initially load testing your application. This may also manifest itself in the client as "503 Server Busy" or "500 Operation Timeout" HTTP status messages from storage operations.

#### **Transient increase in PercentThrottlingError**

If you are seeing spikes in the value of **PercentThrottlingError** that coincide with periods of high activity for the application, you implement an exponential (not linear) back-off strategy for retries in your client. Back-off retries reduce the immediate load on the partition and help your application to smooth out spikes in traffic. For more information about how to implement retry policies using the Storage Client Library, see the [Microsoft.Azure.Storage.RetryPolicies namespace](#).

#### **NOTE**

You may also see spikes in the value of **PercentThrottlingError** that do not coincide with periods of high activity for the application: the most likely cause here is the storage service moving partitions to improve load balancing.

#### **Permanent increase in PercentThrottlingError error**

If you are seeing a consistently high value for **PercentThrottlingError** following a permanent increase in your transaction volumes, or when you are performing your initial load tests on your application, then you need to evaluate how your application is using storage partitions and whether it is approaching the scalability targets for a storage account. For example, if you are seeing throttling errors on a queue (which counts as a single partition), then you should consider using additional queues to spread the transactions across multiple partitions. If you are seeing throttling errors on a table, you need to consider using a different partitioning scheme to spread your transactions across multiple partitions by using a wider range of partition key values. One common cause of this issue is the prepend/append anti-pattern where you select the date as the partition key and then all data on a particular day is written to one partition: under load, this can result in a write bottleneck. Either consider a different partitioning design or evaluate whether using blob storage might be a better solution. Also check whether throttling is occurring as a result of spikes in your traffic and investigate ways of smoothing your pattern of requests.

If you distribute your transactions across multiple partitions, you must still be aware of the scalability limits set for the storage account. For example, if you used ten queues each processing the maximum of 2,000 1KB messages per second, you will be at the overall limit of 20,000 messages per second for the storage account. If you need to process more than 20,000 entities per second, you should consider using multiple storage accounts. You should also bear in mind that the size of your requests and entities has an impact on when the storage service throttles your clients: if you have larger requests and entities, you may be throttled sooner.

Inefficient query design can also cause you to hit the scalability limits for table partitions. For example, a query with a filter that only selects one percent of the entities in a partition but that scans all the entities in a partition will need to access each entity. Every entity read will count towards the total number of transactions in that partition; therefore, you can easily reach the scalability targets.

#### **NOTE**

Your performance testing should reveal any inefficient query designs in your application.

#### **Metrics show an increase in PercentTimeoutError**

Your metrics show an increase in **PercentTimeoutError** for one of your storage services. At the same time, the client receives a high volume of "500 Operation Timeout" HTTP status messages from storage operations.

#### **NOTE**

You may see timeout errors temporarily as the storage service load balances requests by moving a partition to a new server.

The **PercentTimeoutError** metric is an aggregation of the following metrics: **ClientTimeoutError**, **AnonymousClientTimeoutError**, **SASClientTimeoutError**, **ServerTimeoutError**, **AnonymousServerTimeoutError**, and **SASServerTimeoutError**.

The server timeouts are caused by an error on the server. The client timeouts happen because an operation on the server has exceeded the timeout specified by the client; for example, a client using the Storage Client Library can set a timeout for an operation by using the **ServerTimeout** property of the **QueueRequestOptions** class.

Server timeouts indicate a problem with the storage service that requires further investigation. You can use metrics to see if you are hitting the scalability limits for the service and to identify any spikes in traffic that might be causing this problem. If the problem is intermittent, it may be due to load-balancing activity in the service. If the problem is persistent and is not caused by your application hitting the scalability limits of the service, you should raise a support issue. For client timeouts, you must decide if the timeout is set to an appropriate value in the client and either change the timeout value set in the client or investigate how you can improve the performance of the operations in the storage service, for example by optimizing your table queries or reducing the size of your messages.

#### Metrics show an increase in PercentNetworkError

Your metrics show an increase in **PercentNetworkError** for one of your storage services. The **PercentNetworkError** metric is an aggregation of the following metrics: **NetworkError**, **AnonymousNetworkError**, and **SASNetworkError**. These occur when the storage service detects a network error when the client makes a storage request.

The most common cause of this error is a client disconnecting before a timeout expires in the storage service. Investigate the code in your client to understand why and when the client disconnects from the storage service. You can also use Wireshark, or Tcping to investigate network connectivity issues from the client. These tools are described in the [Appendices](#).

#### The client is receiving HTTP 403 (Forbidden) messages

If your client application is throwing HTTP 403 (Forbidden) errors, a likely cause is that the client is using an expired Shared Access Signature (SAS) when it sends a storage request (although other possible causes include clock skew, invalid keys, and empty headers). If an expired SAS key is the cause, you will not see any entries in the server-side Storage Logging log data. The following table shows a sample from the client-side log generated by the Storage Client Library that illustrates this issue occurring:

SOURCE	VERBOSITY	VERBOSITY	CLIENT REQUEST ID	OPERATION TEXT
Microsoft.Azure.Storage	Information	3	85d077ab-...	Starting operation with location Primary per location mode PrimaryOnly.
Microsoft.Azure.Storage	Information	3	85d077ab -...	Starting synchronous request to <a href="https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Synchronous_and_Asynchronous_Requests#Synchronous_request">https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Synchronous_and_Asynchronous_Requests#Synchronous_request</a>
Microsoft.Azure.Storage	Information	3	85d077ab -...	Waiting for response.
Microsoft.Azure.Storage	Warning	2	85d077ab -...	Exception thrown while waiting for response: The remote server returned an error: (403) Forbidden.

Source	Verbosity	Verbosity	Client Request ID	Operation Text
Microsoft.Azure.Storage	Information	3	85d077ab -...	Response received. Status code = 403, Request ID = 9d67c64a-64ed- 4b0d-9515- 3b14bbcd63d, Content-MD5 = , ETag = .
Microsoft.Azure.Storage	Warning	2	85d077ab -...	Exception thrown during the operation: The remote server returned an error: (403) Forbidden..
Microsoft.Azure.Storage	Information	3	85d077ab -...	Checking if the operation should be retried. Retry count = 0, HTTP status code = 403, Exception = The remote server returned an error: (403) Forbidden..
Microsoft.Azure.Storage	Information	3	85d077ab -...	The next location has been set to Primary, based on the location mode.
Microsoft.Azure.Storage	Error	1	85d077ab -...	Retry policy did not allow for a retry. Failing with The remote server returned an error: (403) Forbidden.

In this scenario, you should investigate why the SAS token is expiring before the client sends the token to the server:

- Typically, you should not set a start time when you create a SAS for a client to use immediately. If there are small clock differences between the host generating the SAS using the current time and the storage service, then it is possible for the storage service to receive a SAS that is not yet valid.
- Do not set a very short expiry time on a SAS. Again, small clock differences between the host generating the SAS and the storage service can lead to a SAS apparently expiring earlier than anticipated.
- Does the version parameter in the SAS key (for example `sv=2015-04-05`) match the version of the Storage Client Library you are using? We recommend that you always use the latest version of the [Storage Client Library](#).
- If you regenerate your storage access keys, any existing SAS tokens may be invalidated. This issue may arise if you generate SAS tokens with a long expiry time for client applications to cache.

If you are using the Storage Client Library to generate SAS tokens, then it is easy to build a valid token. However, if you are using the Storage REST API and constructing the SAS tokens by hand, see [Delegating Access with a Shared Access Signature](#).

#### The client is receiving HTTP 404 (Not found) messages

If the client application receives an HTTP 404 (Not found) message from the server, this implies that the object the client was attempting to use (such as an entity, table, blob, container, or queue) does not exist in the storage service. There are a number of possible reasons for this, such as:

- [The client or another process previously deleted the object](#)
- [A Shared Access Signature \(SAS\) authorization issue](#)
- [Client-side JavaScript code does not have permission to access the object](#)
- [Network failure](#)

#### The client or another process previously deleted the object

In scenarios where the client is attempting to read, update, or delete data in a storage service it is usually easy to

identify in the server-side logs a previous operation that deleted the object in question from the storage service. Often, the log data shows that another user or process deleted the object. In the server-side Storage Logging log, the operation-type and requested-object-key columns show when a client deleted an object.

In the scenario where a client is attempting to insert an object, it may not be immediately obvious why this results in an HTTP 404 (Not found) response given that the client is creating a new object. However, if the client is creating a blob it must be able to find the blob container; if the client is creating a message it must be able to find a queue, and if the client is adding a row it must be able to find the table.

You can use the client-side log from the Storage Client Library to gain a more detailed understanding of when the client sends specific requests to the storage service.

The following client-side log generated by the Storage Client library illustrates the problem when the client cannot find the container for the blob it is creating. This log includes details of the following storage operations:

REQUEST ID	OPERATION
07b26a5d...	<b>DeleteIfExists</b> method to delete the blob container. Note that this operation includes a <b>HEAD</b> request to check for the existence of the container.
e2d06d78...	<b>CreateIfNotExists</b> method to create the blob container. Note that this operation includes a <b>HEAD</b> request that checks for the existence of the container. The <b>HEAD</b> returns a 404 message but continues.
de8b1c3c...	<b>UploadFromStream</b> method to create the blob. The <b>PUT</b> request fails with a 404 message

Log entries:

REQUEST ID	OPERATION TEXT
07b26a5d...	Starting synchronous request to <code>https://domemaildist.blob.core.windows.net/azuremmblobcontainer</code>
07b26a5d...	StringToSign = HEAD.....x-ms-client-request-id:07b26a5d-...x-ms-date:Tue, 03 Jun 2014 10:33:11 GMTx-ms-version:2014-02-14./domemaildist/azuremmblobcontainer.restype:container.
07b26a5d...	Waiting for response.
07b26a5d...	Response received. Status code = 200, Request ID = eeead849-...Content-MD5 = , ETag = "0x8D14D2DC63D059B".
07b26a5d...	Response headers were processed successfully, proceeding with the rest of the operation.
07b26a5d...	Downloading response body.
07b26a5d...	Operation completed successfully.
07b26a5d...	Starting synchronous request to <code>https://domemaildist.blob.core.windows.net/azuremmblobcontainer</code>
07b26a5d...	StringToSign = DELETE.....x-ms-client-request-id:07b26a5d-...x-ms-date:Tue, 03 Jun 2014 10:33:12 GMTx-ms-version:2014-02-14./domemaildist/azuremmblobcontainer.restype:container.
07b26a5d...	Waiting for response.
07b26a5d...	Response received. Status code = 202, Request ID = 6ab2a4cf..., Content-MD5 = , ETag = .

REQUEST ID	OPERATION TEXT
07b26a5d-...	Response headers were processed successfully, proceeding with the rest of the operation.
07b26a5d-...	Downloading response body.
07b26a5d-...	Operation completed successfully.
e2d06d78-...	Starting asynchronous request to <code>https://domemaildist.blob.core.windows.net/azuremmblobcontainer</code>
e2d06d78-...	StringToSign = HEAD.....x-ms-client-request-id:e2d06d78-....x-ms-date:Tue, 03 Jun 2014 10:33:12 GMTx-ms-version:2014-02-14./domemaildist/azuremmblobcontainer.restype:container.
e2d06d78-...	Waiting for response.
de8b1c3c-...	Starting synchronous request to <code>https://domemaildist.blob.core.windows.net/azuremmblobcontainer/blobCreate</code>
de8b1c3c-...	StringToSign = PUT..64.qCmF+TQLPhq/YYK50mP9ZQ=.....x-ms-blob-type:BlockBlob.x-ms-client-request-id:de8b1c3c-....x-ms-date:Tue, 03 Jun 2014 10:33:12 GMTx-ms-version:2014-02-14./domemaildist/azuremmblobcontainer/blobCreated.txt.
de8b1c3c-...	Preparing to write request data.
e2d06d78-...	Exception thrown while waiting for response: The remote server returned an error: (404) Not Found..
e2d06d78-...	Response received. Status code = 404, Request ID = 353ae3bc-..., Content-MD5 = , ETag = .
e2d06d78-...	Response headers were processed successfully, proceeding with the rest of the operation.
e2d06d78-...	Downloading response body.
e2d06d78-...	Operation completed successfully.
e2d06d78-...	Starting asynchronous request to <code>https://domemaildist.blob.core.windows.net/azuremmblobcontainer</code>
e2d06d78-...	StringToSign = PUT..0.....x-ms-client-request-id:e2d06d78-....x-ms-date:Tue, 03 Jun 2014 10:33:12 GMTx-ms-version:2014-02-14./domemaildist/azuremmblobcontainer.restype:container.
e2d06d78-...	Waiting for response.
de8b1c3c-...	Writing request data.
de8b1c3c-...	Waiting for response.
e2d06d78-...	Exception thrown while waiting for response: The remote server returned an error: (409) Conflict..
e2d06d78-...	Response received. Status code = 409, Request ID = c27da20e-..., Content-MD5 = , ETag = .

REQUEST ID	OPERATION TEXT
e2d06d78-...	Downloading error response body.
de8b1c3c-...	Exception thrown while waiting for response: The remote server returned an error: (404) Not Found..
de8b1c3c-...	Response received. Status code = 404, Request ID = 0eaeb3e-..., Content-MD5 = , ETag = .
de8b1c3c-...	Exception thrown during the operation: The remote server returned an error: (404) Not Found..
de8b1c3c-...	Retry policy did not allow for a retry. Failing with The remote server returned an error: (404) Not Found..
e2d06d78-...	Retry policy did not allow for a retry. Failing with The remote server returned an error: (409) Conflict..

In this example, the log shows that the client is interleaving requests from the **CreateIfNotExist** method (request ID e2d06d78...) with the requests from the **UploadFromStream** method (de8b1c3c-...). This interleaving happens because the client application is invoking these methods asynchronously. Modify the asynchronous code in the client to ensure that it creates the container before attempting to upload any data to a blob in that container. Ideally, you should create all your containers in advance.

#### A Shared Access Signature (SAS) authorization issue

If the client application attempts to use a SAS key that does not include the necessary permissions for the operation, the storage service returns an HTTP 404 (Not found) message to the client. At the same time, you will also see a non-zero value for **SASAuthorizationError** in the metrics.

The following table shows a sample server-side log message from the Storage Logging log file:

NAME	VALUE
Request start time	2014-05-30T06:17:48.4473697Z
Operation type	GetBlobProperties
Request status	SASAuthorizationError
HTTP status code	404
Authentication type	Sas
Service type	Blob
Request URL	<a href="https://domemaildist.blob.core.windows.net/azureimblobcontainer/blobCreate">https://domemaildist.blob.core.windows.net/azureimblobcontainer/blobCreate</a>
	?sv=2014-02-14&sr=c&si=mypolicy&sig=XXXXX&api-version=2014-02-14
Request ID header	a1f348d5-8032-4912-93ef-b393e5252a3b
Client request ID	2d064953-8436-4ee0-aa0c-65cb874f7929

Investigate why your client application is attempting to perform an operation for which it has not been granted permissions.

#### Client-side JavaScript code does not have permission to access the object

If you are using a JavaScript client and the storage service is returning HTTP 404 messages, you check for the following JavaScript errors in the browser:

```
SEC7120: Origin http://localhost:56309 not found in Access-Control-Allow-Origin header.
SCRIPT7002: XMLHttpRequest: Network Error 0x80070005, Access is denied.
```

#### NOTE

You can use the F12 Developer Tools in Internet Explorer to trace the messages exchanged between the browser and the storage service when you are troubleshooting client-side JavaScript issues.

These errors occur because the web browser implements the [same origin policy](#) security restriction that prevents a web page from calling an API in a different domain from the domain the page comes from.

To work around the JavaScript issue, you can configure Cross Origin Resource Sharing (CORS) for the storage service the client is accessing. For more information, see [Cross-Origin Resource Sharing \(CORS\) Support for Azure Storage Services](#).

The following code sample shows how to configure your blob service to allow JavaScript running in the Contoso domain to access a blob in your blob storage service:

- [.NET v12 SDK](#)
- [.NET v11 SDK](#)

```
var connectionString = Constants.connectionString;

BlobServiceClient blobServiceClient = new BlobServiceClient(connectionString);

BlobServiceProperties sp = blobServiceClient.GetProperties();

// Set the service properties.
sp.DefaultServiceVersion = "2013-08-15";
BlobCorsRule bcr = new BlobCorsRule();
bcr.AllowedHeaders = "*";

bcr.AllowedMethods = "GET,POST";
bcr.AllowedOrigins = "http://www.contoso.com";
bcr.ExposedHeaders = "x-ms-*";
bcr.MaxAgeInSeconds = 5;
sp.Cors.Clear();
sp.Cors.Add(bcr);
blobServiceClient.SetProperties(sp);
```

#### Network Failure

In some circumstances, lost network packets can lead to the storage service returning HTTP 404 messages to the client. For example, when your client application is deleting an entity from the table service you see the client throw a storage exception reporting an "HTTP 404 (Not Found)" status message from the table service. When you investigate the table in the table storage service, you see that the service did delete the entity as requested.

The exception details in the client include the request ID (7e84f12d...) assigned by the table service for the request: you can use this information to locate the request details in the server-side storage logs by searching in the **request-id-header** column in the log file. You could also use the metrics to identify when failures such as this occur and then search the log files based on the time the metrics recorded this error. This log entry shows that the delete failed with an "HTTP (404) Client Other Error" status message. The same log entry also includes the request ID generated by the client in the **client-request-id** column (813ea74f...).

The server-side log also includes another entry with the same **client-request-id** value (813ea74f...) for a successful delete operation for the same entity, and from the same client. This successful delete operation took place very shortly before the failed delete request.

The most likely cause of this scenario is that the client sent a delete request for the entity to the table service, which succeeded, but did not receive an acknowledgment from the server (perhaps due to a temporary network issue). The client then automatically retried the operation (using the same **client-request-id**), and this retry failed because the entity had already been deleted.

If this problem occurs frequently, you should investigate why the client is failing to receive acknowledgments from the table service. If the problem is intermittent, you should trap the "HTTP (404) Not Found" error and log it in the client, but allow the client to continue.

#### The client is receiving HTTP 409 (Conflict) messages

The following table shows an extract from the server-side log for two client operations: **DeleteIfExists** followed immediately by **CreateIfNotExist** using the same blob container name. Each client operation results in two requests sent to the server, first a **GetContainerProperties** request to check if the container exists, followed by

the `DeleteContainer` or `CreateContainer` request.

Timestamp	Operation	Result	Container Name	Client Request ID
05:10:13.7167225	GetContainerProperties	200	mmcont	c9f52c89-...
05:10:13.8167325	DeleteContainer	202	mmcont	c9f52c89-...
05:10:13.8987407	GetContainerProperties	404	mmcont	bc881924-...
05:10:14.2147723	CreateContainer	409	mmcont	bc881924-...

The code in the client application deletes and then immediately recreates a blob container using the same name: the `CreateIfNotExists` method (Client request ID bc881924...) eventually fails with the HTTP 409 (Conflict) error. When a client deletes blob containers, tables, or queues there is a brief period before the name becomes available again.

The client application should use unique container names whenever it creates new containers if the delete/recreate pattern is common.

#### Metrics show low PercentSuccess or analytics log entries have operations with transaction status of ClientOtherErrors

The `PercentSuccess` metric captures the percent of operations that were successful based on their HTTP Status Code. Operations with status codes of 2XX count as successful, whereas operations with status codes in 3XX, 4XX and 5XX ranges are counted as unsuccessful and lower the `PercentSuccess` metric value. In the server-side storage log files, these operations are recorded with a transaction status of `ClientOtherErrors`.

It is important to note that these operations have completed successfully and therefore do not affect other metrics such as availability. Some examples of operations that execute successfully but that can result in unsuccessful HTTP status codes include:

- **ResourceNotFound** (Not Found 404), for example from a GET request to a blob that does not exist.
- **ResourceAlreadyExists** (Conflict 409), for example from a `CreateIfNotExist` operation where the resource already exists.
- **ConditionNotMet** (Not Modified 304), for example from a conditional operation such as when a client sends an `ETag` value and an HTTP `If-None-Match` header to request an image only if it has been updated since the last operation.

You can find a list of common REST API error codes that the storage services return on the page [Common REST API Error Codes](#).

#### Capacity metrics show an unexpected increase in storage capacity usage

If you see sudden, unexpected changes in capacity usage in your storage account, you can investigate the reasons by first looking at your availability metrics; for example, an increase in the number of failed delete requests might lead to an increase in the amount of blob storage you are using as application-specific cleanup operations you might have expected to be freeing up space may not be working as expected (for example, because the SAS tokens used for freeing up space have expired).

#### Your issue arises from using the storage emulator for development or test

You typically use the storage emulator during development and test to avoid the requirement for an Azure storage account. The common issues that can occur when you are using the storage emulator are:

- [Feature "X" is not working in the storage emulator](#)
- [Error "The value for one of the HTTP headers is not in the correct format" when using the storage emulator](#)
- [Running the storage emulator requires administrative privileges](#)

#### Feature "X" is not working in the storage emulator

The storage emulator does not support all of the features of the Azure storage services such as the file service. For more information, see [Use the Azure Storage Emulator for Development and Testing](#).

For those features that the storage emulator does not support, use the Azure storage service in the cloud.

#### Error "The value for one of the HTTP headers is not in the correct format" when using the storage emulator

You are testing your application that uses the Storage Client Library against the local storage emulator and method calls such as `CreateIfNotExists` fail with the error message "The value for one of the HTTP headers is

not in the correct format." This indicates that the version of the storage emulator you are using does not support the version of the storage client library you are using. The Storage Client Library adds the header **x-ms-version** to all the requests it makes. If the storage emulator does not recognize the value in the **x-ms-version** header, it rejects the request.

You can use the Storage Library Client logs to see the value of the **x-ms-version** header it is sending. You can also see the value of the **x-ms-version** header if you use Fiddler to trace the requests from your client application.

This scenario typically occurs if you install and use the latest version of the Storage Client Library without updating the storage emulator. You should either install the latest version of the storage emulator, or use cloud storage instead of the emulator for development and test.

#### **Running the storage emulator requires administrative privileges**

You are prompted for administrator credentials when you run the storage emulator. This only occurs when you are initializing the storage emulator for the first time. After you have initialized the storage emulator, you do not need administrative privileges to run it again.

For more information, see [Use the Azure Storage Emulator for Development and Testing](#). You can also initialize the storage emulator in Visual Studio, which will also require administrative privileges.

#### **You are encountering problems installing the Azure SDK for .NET**

When you try to install the SDK, it fails trying to install the storage emulator on your local machine. The installation log contains one of the following messages:

- CAQuietExec: Error: Unable to access SQL instance
- CAQuietExec: Error: Unable to create database

The cause is an issue with existing LocalDB installation. By default, the storage emulator uses LocalDB to persist data when it simulates the Azure storage services. You can reset your LocalDB instance by running the following commands in a command-prompt window before trying to install the SDK.

```
sqllocaldb stop v11.0
sqllocaldb delete v11.0
delete %USERPROFILE%\WAStorageEmulatorDb3.*
sqllocaldb create v11.0
```

The **delete** command removes any old database files from previous installations of the storage emulator.

#### **You have a different issue with a storage service**

If the previous troubleshooting sections do not include the issue you are having with a storage service, you should adopt the following approach to diagnosing and troubleshooting your issue.

- Check your metrics to see if there is any change from your expected base-line behavior. From the metrics, you may be able to determine whether the issue is transient or permanent, and which storage operations the issue is affecting.
- You can use the metrics information to help you search your server-side log data for more detailed information about any errors that are occurring. This information may help you troubleshoot and resolve the issue.
- If the information in the server-side logs is not sufficient to troubleshoot the issue successfully, you can use the Storage Client Library client-side logs to investigate the behavior of your client application, and tools such as Fiddler, Wireshark to investigate your network.

For more information about using Fiddler, see "[Appendix 1: Using Fiddler to capture HTTP and HTTPS traffic](#)."

For more information about using Wireshark, see "[Appendix 2: Using Wireshark to capture network traffic](#)."

## **Appendices**

The appendices describe several tools that you may find useful when you are diagnosing and troubleshooting issues with Azure Storage (and other services). These tools are not part of Azure Storage and some are third-party products. As such, the tools discussed in these appendices are not covered by any support agreement you may have with Microsoft Azure or Azure Storage, and therefore as part of your evaluation process you should examine the licensing and support options available from the providers of these tools.

### **Appendix 1: Using Fiddler to capture HTTP and HTTPS traffic**

[Fiddler](#) is a useful tool for analyzing the HTTP and HTTPS traffic between your client application and the Azure

storage service you are using.

**NOTE**

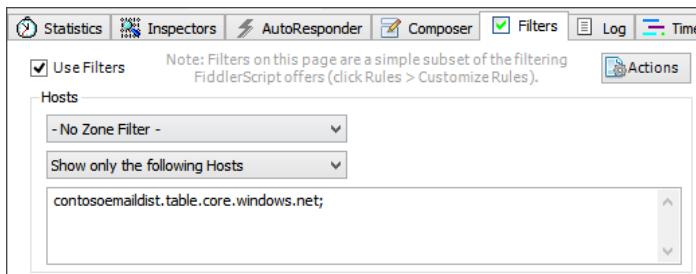
Fiddler can decode HTTPS traffic; you should read the Fiddler documentation carefully to understand how it does this, and to understand the security implications.

This appendix provides a brief walkthrough of how to configure Fiddler to capture traffic between the local machine where you have installed Fiddler and the Azure storage services.

After you have launched Fiddler, it will begin capturing HTTP and HTTPS traffic on your local machine. The following are some useful commands for controlling Fiddler:

- Stop and start capturing traffic. On the main menu, go to **File** and then click **Capture Traffic** to toggle capturing on and off.
- Save captured traffic data. On the main menu, go to **File**, click **Save**, and then click **All Sessions**: this enables you to save the traffic in a Session Archive file. You can reload a Session Archive later for analysis, or send it if requested to Microsoft support.

To limit the amount of traffic that Fiddler captures, you can use filters that you configure in the **Filters** tab. The following screenshot shows a filter that captures only traffic sent to the **contosoemaildist.table.core.windows.net** storage endpoint:

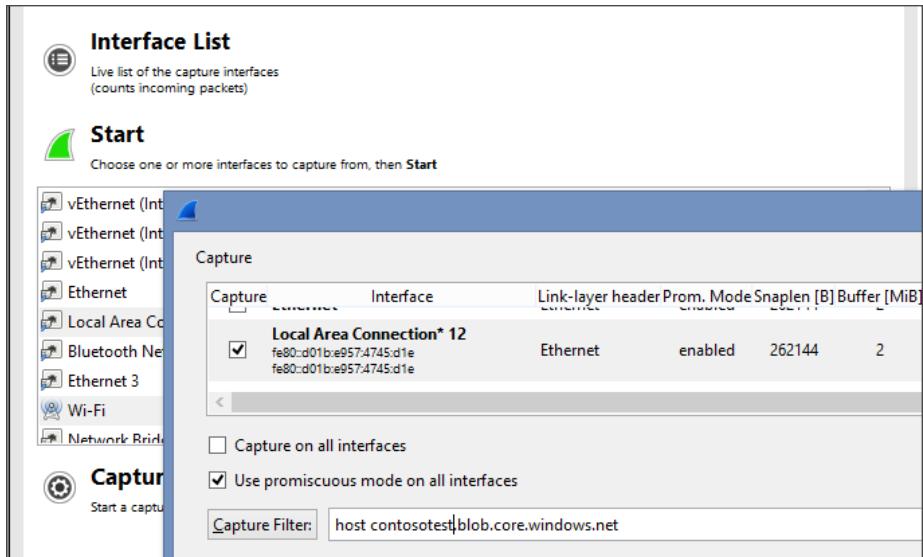


#### Appendix 2: Using Wireshark to capture network traffic

[Wireshark](#) is a network protocol analyzer that enables you to view detailed packet information for a wide range of network protocols.

The following procedure shows you how to capture detailed packet information for traffic from the local machine where you installed Wireshark to the table service in your Azure storage account.

1. Launch Wireshark on your local machine.
2. In the **Start** section, select the local network interface or interfaces that are connected to the internet.
3. Click **Capture Options**.
4. Add a filter to the **Capture Filter** textbox. For example, **host contosoemaildist.table.core.windows.net** will configure Wireshark to capture only packets sent to or from the table service endpoint in the **contosoemaildist** storage account. Check out the [complete list of Capture Filters](#).



5. Click **Start**. Wireshark will now capture all the packets send to or from the table service endpoint as you use your client application on your local machine.
6. When you have finished, on the main menu click **Capture** and then **Stop**.
7. To save the captured data in a Wireshark Capture File, on the main menu click **File** and then **Save**.

WireShark will highlight any errors that exist in the **packetlist** window. You can also use the **Expert Info** window (click **Analyze**, then **Expert Info**) to view a summary of errors and warnings.

Category	Count
Errors	3 (3)
Warnings	5 (6)
Notes	7 (32)
Chats	13 (24)
Details	65
Packet Comments	0

You can also choose to view the TCP data as the application layer sees it by right-clicking on the TCP data and selecting **Follow TCP Stream**. This is useful if you captured your dump without a capture filter. For more information, see [Following TCP Streams](#).

**NOTE**

For more information about using Wireshark, see the [Wireshark Users Guide](#).

#### **Appendix 4: Using Excel to view metrics and log data**

Many tools enable you to download the Storage Metrics data from Azure table storage in a delimited format that makes it easy to load the data into Excel for viewing and analysis. Storage Logging data from Azure Blob Storage is already in a delimited format that you can load into Excel. However, you will need to add appropriate column headings based in the information at [Storage Analytics Log Format](#) and [Storage Analytics Metrics Table Schema](#).

To import your Storage Logging data into Excel after you download it from blob storage:

- On the **Data** menu, click **From Text**.
- Browse to the log file you want to view and click **Import**.
- On step 1 of the **Text Import Wizard**, select **Delimited**.

On step 1 of the **Text Import Wizard**, select **Semicolon** as the only delimiter and choose double-quote as the **Text qualifier**. Then click **Finish** and choose where to place the data in your workbook.

#### **Appendix 5: Monitoring with Application Insights for Azure DevOps**

You can also use the Application Insights feature for Azure DevOps as part of your performance and availability monitoring. This tool can:

- Make sure your web service is available and responsive. Whether your app is a web site or a device app that uses a web service, it can test your URL every few minutes from locations around the world, and let you know if there's a problem.
- Quickly diagnose any performance issues or exceptions in your web service. Find out if CPU or other resources are being stretched, get stack traces from exceptions, and easily search through log traces. If the app's performance drops below acceptable limits, Microsoft can send you an email. You can monitor both .NET and Java web services.

You can find more information at [What is Application Insights](#).

## Next steps

For more information about analytics in Azure Storage, see these resources:

- [Monitor a storage account in the Azure portal](#)
- [Storage analytics](#)
- [Storage analytics metrics](#)
- [Storage analytics metrics table schema](#)
- [Storage analytics logs](#)
- [Storage analytics log format](#)

# Create a storage account to use with Azure Data Lake Storage Gen2

8/22/2022 • 2 minutes to read • [Edit Online](#)

To use Data Lake Storage Gen2 capabilities, create a storage account that has a hierarchical namespace.

For step-by-step guidance, see [Create a storage account](#).

As you create the account, make sure to select the options described in this article.

## Choose a storage account type

Data Lake Storage capabilities are supported in the following types of storage accounts:

- Standard general-purpose v2
- Premium block blob

For information about how to choose between them, see [storage account overview](#).

You can choose between these two types of accounts in the **Basics** tab of the [Create a storage account](#) page.

To create a standard general-purpose v2 account, select **Standard**.

To create a premium block blob account, select **Premium**. Then, in the **Premium account type** dropdown list, select **Block blobs**.

### Instance details

If you need to create a legacy storage account type, please click [here](#).

The screenshot shows the 'Create a storage account' Basics tab. It includes fields for 'Storage account name' (with a red asterisk), 'Region' set to '(US) East US', 'Performance' (radio button for 'Premium' selected, highlighted with a red border), and a dropdown for 'Premium account type' showing 'Block blobs' (also highlighted with a red border). A callout box over the dropdown details 'Block blobs: Best for high transaction rates or low storage latency'. At the bottom are 'Review + create' and 'NEXT > Advanced <' buttons.

Storage account name ⓘ \*

Region ⓘ \*

(US) East US

Performance ⓘ \*

Standard: Recommended for most scenarios (general-purpose v2 account)

Premium: Recommended for scenarios that require low latency.

Premium account type ⓘ \*

Block blobs

Block blobs:  
Best for high transaction rates or low storage latency

File shares:  
Best for enterprise or high-performance applications that need to scale

Page blobs:  
Best for random read and write operations

Review + create

NEXT > Advanced <

## Enable the hierarchical namespace

Unlock Data Lake Storage capabilities by selecting the **enable hierarchical namespace** setting in the **Advanced** tab of the [Create storage account](#) page.

The following image shows this setting in the [Create storage account](#) page.

Home > Create a resource >

## Create a storage account

Basics

Advanced

Networking

Data protection

Tags

Review + create

 Certain options have been disabled by default due to the combination of storage account performance, redundancy, and region.

### Security

Configure security settings that impact your storage account.

Enable secure transfer 



Enable infrastructure encryption 



 Sign up is currently required to enable infrastructure encryption on a per-subscription basis. [Sign up](#)

Enable blob public access 



Enable storage account key access 



Minimum TLS version 

Version 1.2



### Data Lake Storage Gen2

The Data Lake Storage Gen2 hierarchical namespace accelerates big data analytics workloads and enables file-level access control lists (ACLs). [Learn more](#)

Enable hierarchical namespace 



To enable Data Lake Storage capabilities on an existing account, see [Upgrade Azure Blob Storage with Azure Data Lake Storage Gen2 capabilities](#).

## Next steps

- [Storage account overview](#)
- [Upgrade Azure Blob Storage with Azure Data Lake Storage Gen2 capabilities](#)
- [Access control in Azure Data Lake Storage Gen2](#)

# Migrate Azure Data Lake Storage from Gen1 to Gen2 by using the Azure portal

8/22/2022 • 10 minutes to read • [Edit Online](#)

On **Feb. 29, 2024** Azure Data Lake Storage Gen1 will be retired. For more information, see the [official announcement](#). If you use Azure Data Lake Storage Gen1, make sure to migrate to Azure Data Lake Storage Gen2 prior to that date.

This article shows you how to simplify the migration by using the Azure portal. You can provide your consent in the Azure portal and then migrate your data and metadata (such as timestamps and ACLs) automatically from Azure Data Lake Storage Gen1 to Azure Data Lake Storage Gen2. For easier reading, this article uses the term *Gen1* to refer to Azure Data Lake Storage Gen1, and the term *Gen2* to refer to Azure Data Lake Storage Gen2.

## NOTE

Your account may not qualify for portal-based migration based on certain constraints. When the **Migrate data** button is not enabled in the Azure portal for your Gen1 account, if you have a support plan, you can [file a support request](#). You can also get answers from community experts in [Microsoft Q&A](#).

## WARNING

Azure Data Lake Storage Gen2 doesn't support Azure Data Lake Analytics. If you're using Azure Data Lake Analytics, you'll need to migrate before proceeding. See [Migrate Azure Data Lake Analytics workloads](#) for more information.

To migrate to Gen2 using the Azure portal, follow these steps:

- ✓ Step 1: Assess readiness
  - Verify RBAC role assignments
  - [Migrate Azure Data Lake Analytics workloads](#) (if any)
- ✓ Step 2: [Create a storage account with Gen2 capabilities](#)
- ✓ Step 3: [Migrate data using the Azure portal](#)
- ✓ Step 4: [Migrate workloads and applications](#)

Before you start, be sure to read the general guidance on how to migrate from Gen1 to Gen2 in [Azure Data Lake Storage migration guidelines and patterns](#).

## Create a storage account with Gen2 capabilities

Azure Data Lake Storage Gen2 is not a dedicated storage account or service type. It's a set of capabilities that you can obtain by enabling the **Hierarchical namespace** feature of an Azure storage account. To create an account that has Gen2 capabilities, see [Create a storage account to use with Azure Data Lake Storage Gen2](#).

As you create the account, make sure to configure settings with the following values.

Setting	Value
Storage account name	Any name that you want. This name doesn't have to match the name of your Gen1 account and can be in any subscription of your choice.
Location	The same region used by the Data Lake Storage Gen1 account
Replication	LRS or ZRS
Minimum TLS version	1.0
NFS v3	Disabled
Hierarchical namespace	Enabled

#### NOTE

The migration tool in the Azure portal doesn't move account settings. Therefore, after you've created the account, you'll have to manually configure settings such as encryption, network firewalls, data protection.

#### IMPORTANT

Ensure that you use a fresh, newly created Gen2 account that has no history of any usage. **Don't** migrate to a previously used Gen2 account or use a Gen2 account in which containers have been deleted to make the Gen2 account empty.

## Verify RBAC role assignments

For Gen2, ensure that the [Storage Blob Data Owner](#) role has been assigned to your Azure Active Directory (Azure AD) user identity in the scope of the storage account, parent resource group, or subscription.

For Gen1, ensure that the [Owner](#) role has been assigned to your Azure AD identity in the scope of the Gen1 account, parent resource group, or subscription.

## Migrate Azure Data Lake Analytics workloads

Azure Data Lake Storage Gen2 doesn't support Azure Data Lake Analytics. Azure Data Lake Analytics [will be retired](#) on February 29, 2024. If you attempt to use the Azure portal to migrate an Azure Data Lake Storage Gen1 account that is used for Azure Data Lake Analytics, it's possible that you'll break your Azure Data Lake Analytics workloads. You must first [migrate your Azure Data Lake Analytics workloads to Azure Synapse Analytics](#) or another supported compute platform before attempting to migrate your Gen1 account.

For more information, see [Manage Azure Data Lake Analytics using the Azure portal](#).

## Perform the migration

Before you begin, review the two migration options below, and decide whether to only copy data from Gen1 to Gen2 (recommended) or perform a complete migration.

#### NOTE

No matter which option you select, a container named **gen1** will be created on the Gen2 account, and all data from the Gen1 account will be copied to this new 'gen1' container. When the migration is complete, in order to find the data on a path that existed on Gen1, you must add the prefix **gen1/** to the same path to access it on Gen2. For example, a path that was named 'FolderRoot/FolderChild/FileName.csv' on Gen1 will be available at 'gen1/FolderRoot/FolderChild/FileName.csv' on Gen2. Container names can't be renamed on Gen2, so this **gen1** container on Gen2 can't be renamed post migration. However, the data can be copied to a new container in Gen2 if needed.

## Choose a migration option

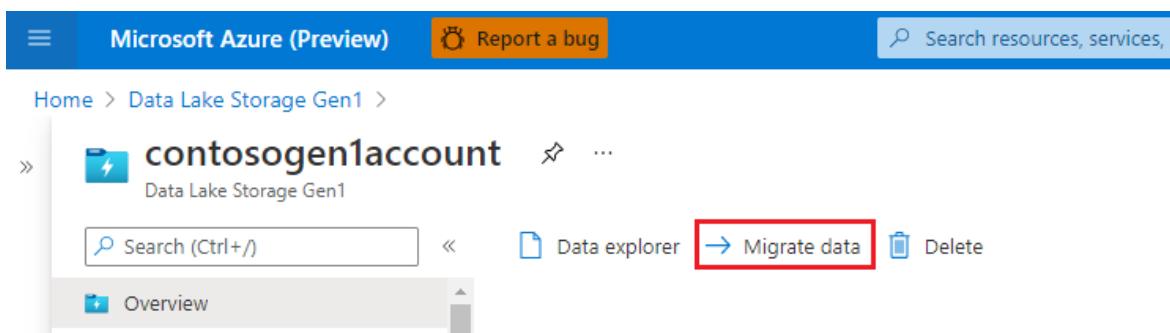
**Option 1: Copy data only (recommended).** In this option, data will be copied from Gen1 to Gen2. As the data is being copied, the Gen1 account will become read-only. After the data is copied, both the Gen1 and Gen2 accounts will be accessible. However, you must update the applications and compute workloads to use the new ADLS Gen2 endpoint.

**Option 2: Perform a complete migration.** In this option, data will be copied from Gen1 to Gen2. After the data is copied, all the traffic from the Gen1 account will be redirected to the Gen2 account. Redirected requests will use the [Gen1 compatibility layer](#) to translate Gen1 API calls to Gen2 equivalents. During the migration, the Gen1 account will become read-only. After the migration is complete, the Gen1 account won't be accessible.

Whichever option you choose, after you've migrated and verified that all your workloads work as expected, you can delete the Gen1 account.

### Option 1: Copy data from Gen1 to Gen2

1. Sign in to the [Azure portal](#) to get started.
2. Locate your Data Lake Storage Gen1 account and display the account overview.
3. Select the **Migrate data** button.



The screenshot shows the Azure portal interface for a Data Lake Storage Gen1 account named 'contosogen1account'. At the top, there's a blue header bar with the Microsoft Azure logo, a 'Report a bug' button, and a search bar. Below the header, the account name 'contosogen1account' is displayed along with its type 'Data Lake Storage Gen1'. A navigation bar includes links for 'Home', 'Data Lake Storage Gen1', 'Search (Ctrl+)', 'Data explorer', 'Migrate data' (which is highlighted with a red box), and 'Delete'. A sidebar on the left shows 'Overview' and other account details. The main content area displays general account information and metrics.

4. Select **Copy data to a new Gen2 account**.



5. Give Microsoft consent to perform the data migration by selecting the checkbox. Then, click the **Apply** button.



#### IMPORTANT

While your data is being migrated, your Gen1 account becomes read-only and your Gen2-enabled account is disabled. When the migration is finished, you can read and write to both accounts.

You can stop the migration at any time by selecting the **Stop migration** button.

## Microsoft Managed Gen1 to Gen2 Migration ...



#### Option 2: Perform a complete migration

1. Sign in to the [Azure portal](#) to get started.
2. Locate your Data Lake Storage Gen1 account and display the account overview.
3. Select the **Migrate data** button.

4. Select **Complete migration to a new Gen2 account**.



5. Give Microsoft consent to perform the data migration by selecting the checkbox. Then, click the **Apply** button.



#### IMPORTANT

While your data is being migrated, your Gen1 account becomes read-only and the Gen2-enabled account is disabled.

Also, while the Gen1 URI is being redirected, both accounts are disabled.

When the migration is finished, your Gen1 account will be disabled. The data in your Gen1 account won't be accessible and will be deleted after 30 days. Your Gen2 account will be available for reads and writes.

You can stop the migration at any time before the URI is redirected by selecting the **Stop migration** button.

[Stop migration](#)[Refresh](#)

## Migrate workloads and applications

1. Configure [services in your workloads](#) to point to your Gen2 endpoint. For links to articles that help you configure Azure Databricks, HDInsight, and other Azure services to use Gen2, see [Azure services that support Azure Data Lake Storage Gen2](#).
2. Update applications to use Gen2 APIs. See these guides:

ENVIRONMENT	ARTICLE
Azure Storage Explorer	<a href="#">Use Azure Storage Explorer to manage directories and files in Azure Data Lake Storage Gen2</a>
.NET	<a href="#">Use .NET to manage directories and files in Azure Data Lake Storage Gen2</a>
Java	<a href="#">Use Java to manage directories and files in Azure Data Lake Storage Gen2</a>
Python	<a href="#">Use Python to manage directories and files in Azure Data Lake Storage Gen2</a>
JavaScript (Node.js)	<a href="#">Use JavaScript SDK in Node.js to manage directories and files in Azure Data Lake Storage Gen2</a>
REST API	<a href="#">Azure Data Lake Store REST API</a>

3. Update scripts to use Data Lake Storage Gen2 [PowerShell cmdlets](#), and [Azure CLI commands](#).
4. Search for URI references that contain the string `adl://` in code files, or in Databricks notebooks, Apache Hive HQL files or any other file used as part of your workloads. Replace these references with the [Gen2 formatted URI](#) of your new storage account. For example: the Gen1 URI:  
`adl://mydatalakestore.azuredatalakestore.net/mydirectory/myfile` might become  
`abfss://myfilesystem@mydatalakestore.dfs.core.windows.net/mydirectory/myfile`.

## Gen1 compatibility layer

This layer attempts to provide application compatibility between Gen1 and Gen2 as a convenience during the migration, so that applications can continue using Gen1 APIs to interact with data in the Gen2-enabled account. This layer has limited functionality and it is strongly advised to validate the workloads with test accounts if you use this approach as part of migration. The compatibility layer runs on the server, so there's nothing to install.

### IMPORTANT

Microsoft does not recommend this capability as a replacement for migrating your workloads and applications. Support for the Gen1 compatibility layer will end when Gen1 is retired on Feb. 29, 2024.

To encounter the least number of issues with the compatibility layer, make sure that your Gen1 SDKs use the following versions (or higher).

LANGUAGE	SDK VERSION
.NET	2.3.9
Java	1.1.21
Python	0.0.51

The following functionality isn't supported in the compatibility layer.

- ListStatus API option to ListBefore an entry.
- ListStatus API with over 4,000 files without a continuation token.
- Chunk-encoding for append operations.
- Any API calls that use <https://management.azure.com/> as the Azure Active Directory (Azure AD) token audience.
- File or directory names with only spaces or tabs, ending with a `.`, containing a `:`, or with multiple consecutive forward slashes (`//`).

## Frequently asked questions

### How much does the data migration cost?

There is no cost to use the portal-based migration tool, however you will be billed for usage of Azure Data Lake Gen1 and Gen2 services. During the data migration, you will be billed for the data storage and transactions of the Gen1 account.

Post migration, if you chose the option that copies only data, then you will be billed for the data storage and transactions for both Azure Data Lake Gen1 and Gen2 accounts. To avoid being billed for the Gen1 account, delete the Gen1 account after you've updated your applications to point to Gen2. If you chose to perform a complete migration, you will be billed only for the data storage and transactions of the Gen2-enabled account.

### While providing consent I encountered the error message *Migration initiation failed*. What should I do next?

Make sure all your Azure Data lake Analytics accounts are [migrated to Azure Synapse Analytics](#) or another supported compute platform. Once Azure Data Lake Analytics accounts are migrated, retry the consent. If you see the issue further and you have a support plan, you can [file a support request](#). You can also get answers from community experts in [Microsoft Q&A](#).

### After the migration completes, can I go back to using the Gen1 account?

If you used [Option 1: Copy data from Gen1 to Gen2](#) mentioned above, then both the Gen1 and Gen2 accounts are available for reads and writes post migration. However, if you used [Option 2: Perform a complete migration](#), then going back to the Gen1 account isn't supported. In Option 2, after the migration completes, the data in your Gen1 account won't be accessible and will be deleted after 30 days. You can continue to view the Gen1 account in the Azure portal, and when you're ready, you can delete the Gen1 account.

### I would like to enable Geo-redundant storage (GRS) on the Gen2 account, how do I do that?

Once the migration is complete, both in "Copy data" and "Complete migration" options, you can go ahead and change the redundancy option to GRS as long as you don't plan to use the application compatibility layer. The application compatibility will not work on accounts that use GRS redundancy.

### Gen1 doesn't have containers and Gen2 has them - what should I expect?

When we copy the data over to your Gen2-enabled account, we automatically create a container named 'Gen1'. In Gen2 container names cannot be renamed and hence post migration data can be copied to new container in Gen2 as needed.

### What should I consider in terms of migration performance?

When you copy the data over to your Gen2-enabled account, two factors that can affect performance are the number of files and the amount of metadata you have. For example, many small files can affect the performance of the migration.

## Next steps

- Learn about migration in general. For more information, see [Migrate Azure Data Lake Storage from Gen1 to Gen2](#).

# Azure Data Lake Storage migration guidelines and patterns

8/22/2022 • 8 minutes to read • [Edit Online](#)

You can migrate your data, workloads, and applications from Azure Data Lake Storage Gen1 to Azure Data Lake Storage Gen2. This article explains the recommended migration approach and covers the different migration patterns and when to use each. For easier reading, this article uses the term *Gen1* to refer to Azure Data Lake Storage Gen1, and the term *Gen2* to refer to Azure Data Lake Storage Gen2.

On **Feb 29, 2024** Azure Data Lake Storage Gen1 will be retired. For more information, see the [official announcement](#). If you use Azure Data Lake Storage Gen1, make sure to migrate to Azure Data Lake Storage Gen2 prior to that date. This article shows you how to do that.

Azure Data Lake Storage Gen2 is built on [Azure Blob storage](#) and provides a set of capabilities dedicated to big data analytics. [Data Lake Storage Gen2](#) combines features from [Azure Data Lake Storage Gen1](#), such as file system semantics, directory, and file level security and scale with low-cost, tiered storage, high availability/disaster recovery capabilities from [Azure Blob storage](#).

## NOTE

Because Gen1 and Gen2 are different services, there is no in-place upgrade experience. To simplify the migration to Gen2 by using the Azure portal, see [Migrate Azure Data Lake Storage from Gen1 to Gen2 by using the Azure portal](#).

## Recommended approach

To migrate from Gen1 to Gen2, we recommend the following approach.

- ✓ Step 1: Assess readiness
- ✓ Step 2: Prepare to migrate
- ✓ Step 3: Migrate data and application workloads
- ✓ Step 4: Cutover from Gen1 to Gen2

### Step 1: Assess readiness

1. Learn about the [Data Lake Storage Gen2 offering](#); its benefits, costs, and general architecture.
2. [Compare the capabilities](#) of Gen1 with those of Gen2.
3. Review a list of [known issues](#) to assess any gaps in functionality.
4. Gen2 supports Blob storage features such as [diagnostic logging](#), [access tiers](#), and [Blob storage lifecycle management policies](#). If you're interesting in using any of these features, review [current level of support](#).
5. Review the current state of [Azure ecosystem support](#) to ensure that Gen2 supports any services that your solutions depend upon.

### Step 2: Prepare to migrate

1. Identify the data sets that you'll migrate.

Take this opportunity to clean up data sets that you no longer use. Unless you plan to migrate all of your data at one time, Take this time to identify logical groups of data that you can migrate in phases.

Perform an [Ageing Analysis](#) (or similar) on your Gen1 account to identify which files or folders stay in inventory for a long time or are perhaps becoming obsolete.

## 2. Determine the impact that a migration will have on your business.

For example, consider whether you can afford any downtime while the migration takes place. These considerations can help you to identify a suitable migration pattern, and to choose the most appropriate tools.

## 3. Create a migration plan.

We recommend these [migration patterns](#). You can choose one of these patterns, combine them together, or design a custom pattern of your own.

## Step 3: Migrate data, workloads, and applications

Migrate data, workloads, and applications by using the pattern that you prefer. We recommend that you validate scenarios incrementally.

1. [Create a storage account](#) and enable the hierarchical namespace feature.
2. Migrate your data.
3. Configure [services in your workloads](#) to point to your Gen2 endpoint.

For HDInsight clusters, you can add storage account configuration settings to the %HADOOP\_HOME%/conf/core-site.xml file. If you plan to migrate external Hive tables from Gen1 to Gen2, then make sure to add storage account settings to the %HIVE\_CONF\_DIR%/hive-site.xml file as well.

You can modify the settings each file by using [Apache Ambari](#). To find storage account settings, see [Hadoop Azure Support: ABFS — Azure Data Lake Storage Gen2](#). This example uses the `fs.azure.account.key` setting to enable Shared Key authorization:

```
<property>
 <name>fs.azure.account.key.abfswales1.dfs.core.windows.net</name>
 <value>your-key-goes-here</value>
</property>
```

For links to articles that help you configure HDInsight, Azure Databricks, and other Azure services to use Gen2, see [Azure services that support Azure Data Lake Storage Gen2](#).

## 4. Update applications to use Gen2 APIs. See these guides:

ENVIRONMENT	ARTICLE
Azure Storage Explorer	<a href="#">Use Azure Storage Explorer to manage directories and files in Azure Data Lake Storage Gen2</a>
.NET	<a href="#">Use .NET to manage directories and files in Azure Data Lake Storage Gen2</a>
Java	<a href="#">Use Java to manage directories and files in Azure Data Lake Storage Gen2</a>
Python	<a href="#">Use Python to manage directories and files in Azure Data Lake Storage Gen2</a>

ENVIRONMENT	ARTICLE
JavaScript (Node.js)	<a href="#">Use JavaScript SDK in Node.js to manage directories and files in Azure Data Lake Storage Gen2</a>
REST API	<a href="#">Azure Data Lake Store REST API</a>

5. Update scripts to use Data Lake Storage Gen2 [PowerShell cmdlets](#), and [Azure CLI commands](#).

6. Search for URI references that contain the string `adl://` in code files, or in Databricks notebooks, Apache Hive HQL files or any other file used as part of your workloads. Replace these references with the [Gen2 formatted URI](#) of your new storage account. For example: the Gen1 URI:

`adl://mydatalakestore.azuredatalakestore.net/mydirectory/myfile` might become  
`abfss://myfilesystem@mydatalakestore.dfs.core.windows.net/mydirectory/myfile`.

7. Configure the security on your account to include [Azure roles, file and folder level security](#), and [Azure Storage firewalls and virtual networks](#).

#### Step 4: Cutover from Gen1 to Gen2

After you're confident that your applications and workloads are stable on Gen2, you can begin using Gen2 to satisfy your business scenarios. Turn off any remaining pipelines that are running on Gen1 and decommission your Gen1 account.

## Gen1 vs Gen2 capabilities

This table compares the capabilities of Gen1 to that of Gen2.

AREA	GEN1	GEN2
Data organization	Hierarchical namespace File and folder support	Hierarchical namespace Container, file and folder support
Geo-redundancy	LRS	LRS, ZRS, GRS, RA-GRS
Authentication	Azure Active Directory (Azure AD) managed identity Service principals	Azure AD managed identity Service principals Shared Access Key
Authorization	Management - <a href="#">Azure RBAC</a> Data - <a href="#">ACLs</a>	Management - <a href="#">Azure RBAC</a> Data - <a href="#">ACLs</a> , <a href="#">Azure RBAC</a>
Encryption - Data at rest	Server side - with Microsoft-managed or <a href="#">customer-managed</a> keys	Server side - with <a href="#">Microsoft-managed</a> or <a href="#">customer-managed</a> keys
VNET Support	<a href="#">VNET Integration</a>	<a href="#">Service Endpoints</a> , <a href="#">Private Endpoints</a>
Developer experience	<a href="#">REST</a> , <a href="#">.NET</a> , <a href="#">Java</a> , <a href="#">Python</a> , <a href="#">PowerShell</a> , <a href="#">Azure CLI</a>	Generally available - <a href="#">REST</a> , <a href="#">.NET</a> , <a href="#">Java</a> , <a href="#">Python</a> Public preview - <a href="#">JavaScript</a> , <a href="#">PowerShell</a> , <a href="#">Azure CLI</a>
Resource logs	Classic logs <a href="#">Azure Monitor integrated</a>	<a href="#">Classic logs</a> - Generally available <a href="#">Azure Monitor integrated</a> - Preview

Area	Gen1	Gen2
Ecosystem	HDInsight (3.6), Azure Databricks (3.1 and above), Azure Synapse Analytics, ADF	HDInsight (3.6, 4.0), Azure Databricks (5.1 and above), Azure Synapse Analytics, ADF

## Gen1 to Gen2 patterns

Choose a migration pattern, and then modify that pattern as needed.

Migration Pattern	Details
Lift and Shift	The simplest pattern. Ideal if your data pipelines can afford downtime.
Incremental copy	Similar to <i>lift and shift</i> , but with less downtime. Ideal for large amounts of data that take longer to copy.
Dual pipeline	Ideal for pipelines that can't afford any downtime.
Bidirectional sync	Similar to <i>dual pipeline</i> , but with a more phased approach that is suited for more complicated pipelines.

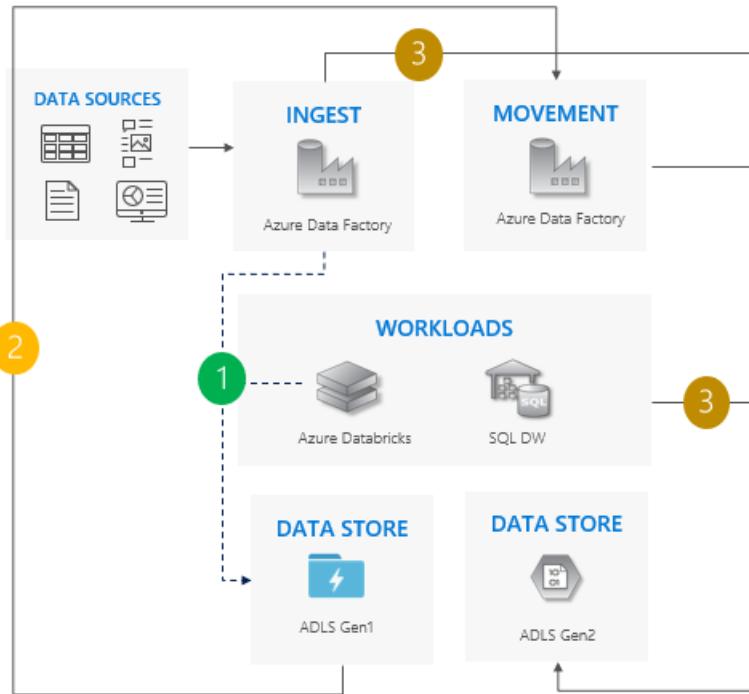
Let's take a closer look at each pattern.

### Lift and shift pattern

This is the simplest pattern.

1. Stop all writes to Gen1.
2. Move data from Gen1 to Gen2. We recommend [Azure Data Factory](#) or by using the [Azure portal](#). ACLs copy with the data.
3. Point ingest operations and workloads to Gen2.
4. Decommission Gen1.

Check out our sample code for the lift and shift pattern in our [Lift and Shift migration sample](#).



#### Considerations for using the lift and shift pattern

- ✓ Cutover from Gen1 to Gen2 for all workloads at the same time.
- ✓ Expect downtime during the migration and the cutover period.
- ✓ Ideal for pipelines that can afford downtime and all apps can be upgraded at one time.

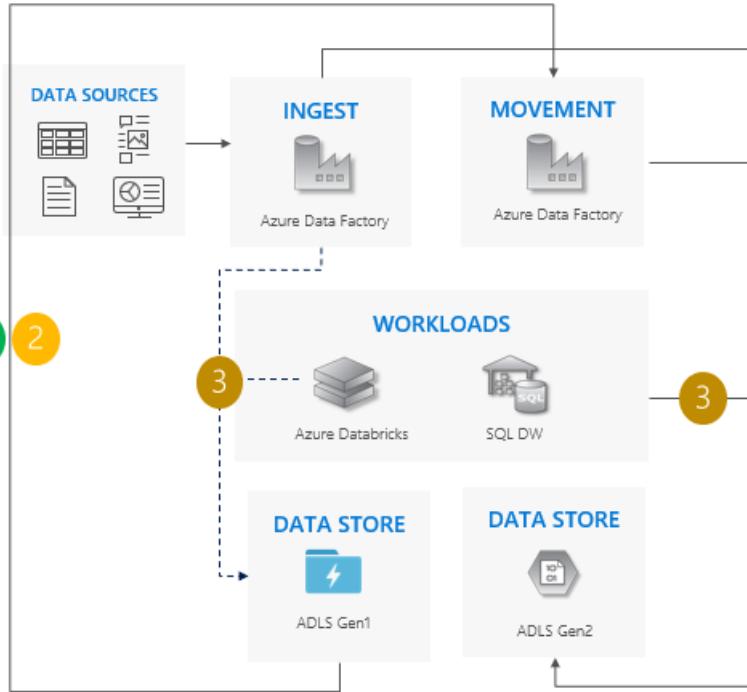
#### TIP

Consider using the [Azure portal](#) to shorten downtime and reduce the number of steps required by you to complete the migration.

#### Incremental copy pattern

1. Start moving data from Gen1 to Gen2. We recommend [Azure Data Factory](#). ACLs copy with the data.
2. Incrementally copy new data from Gen1.
3. After all data is copied, stop all writes to Gen1, and point workloads to Gen2.
4. Decommission Gen1.

Check out our sample code for the incremental copy pattern in our [Incremental copy migration sample](#).



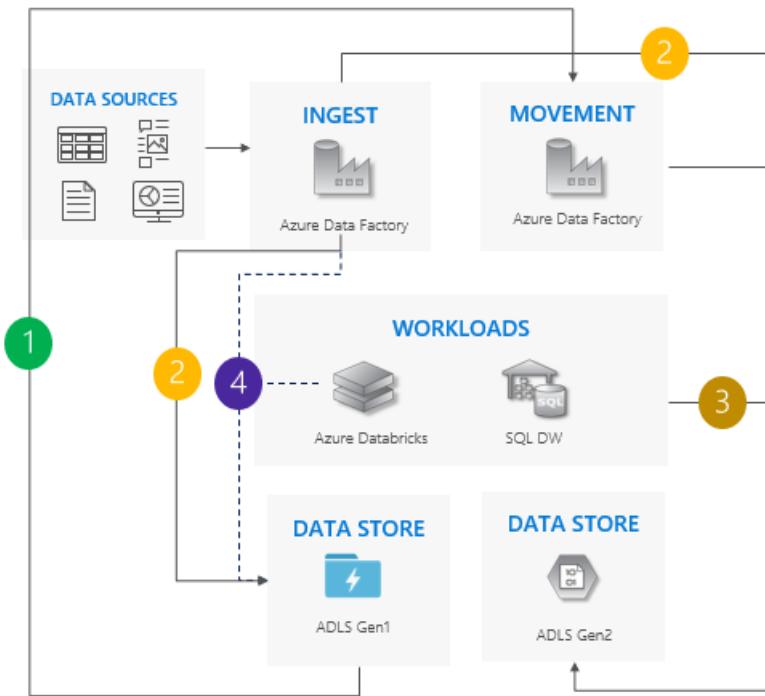
#### Considerations for using the incremental copy pattern:

- ✓ Cutover from Gen1 to Gen2 for all workloads at the same time.
- ✓ Expect downtime during cutover period only.
- ✓ Ideal for pipelines where all apps upgraded at one time, but the data copy requires more time.

#### Dual pipeline pattern

1. Move data from Gen1 to Gen2. We recommend [Azure Data Factory](#). ACLs copy with the data.
2. Ingest new data to both Gen1 and Gen2.
3. Point workloads to Gen2.
4. Stop all writes to Gen1 and then decommission Gen1.

Check out our sample code for the dual pipeline pattern in our [Dual Pipeline migration sample](#).



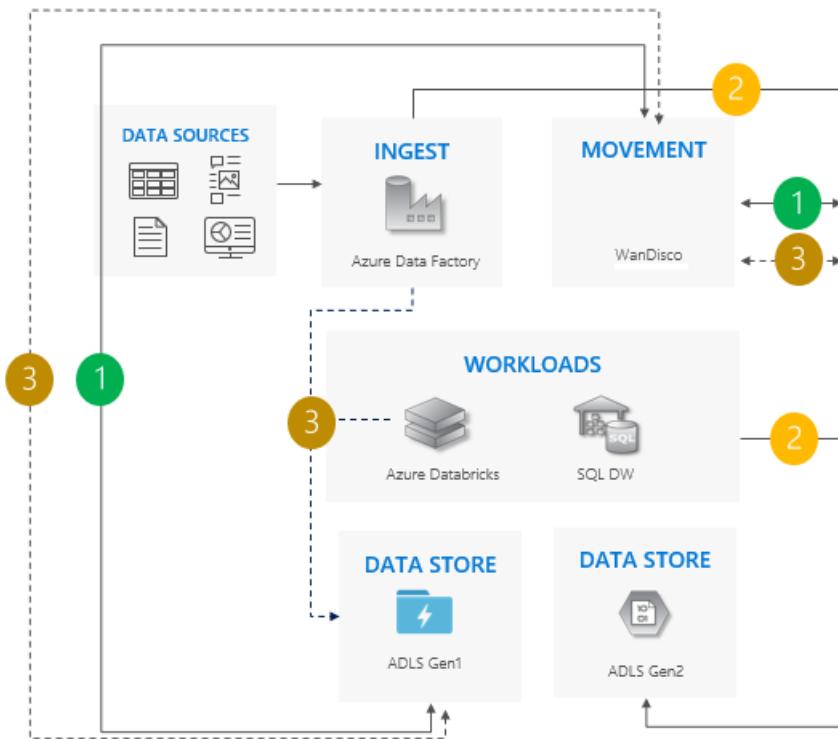
#### Considerations for using the dual pipeline pattern:

- ✓ Gen1 and Gen2 pipelines run side-by-side.
- ✓ Supports zero downtime.
- ✓ Ideal in situations where your workloads and applications can't afford any downtime, and you can ingest into both storage accounts.

#### Bi-directional sync pattern

1. Set up bidirectional replication between Gen1 and Gen2. We recommend [WanDisco](#). It offers a repair feature for existing data.
2. When all moves are complete, stop all writes to Gen1 and turn off bidirectional replication.
3. Decommission Gen1.

Check out our sample code for the bidirectional sync pattern in our [Bidirectional Sync migration sample](#).



#### Considerations for using the bi-directional sync pattern:

- ✓ Ideal for complex scenarios that involve a large number of pipelines and dependencies where a phased approach might make more sense.
- ✓ Migration effort is high, but it provides side-by-side support for Gen1 and Gen2.

## Next steps

- Learn about the various parts of setting up security for a storage account. For more information, see [Azure Storage security guide](#).
- Optimize the performance for your Data Lake Store. See [Optimize Azure Data Lake Storage Gen2 for performance](#)
- Review the best practices for managing your Data Lake Store. See [Best practices for using Azure Data Lake Storage Gen2](#)

## See also

- [The Hitchhiker's Guide to the Data Lake](#)
- [Access control model in Azure Data Lake Storage Gen2](#)

# Migrate from on-prem HDFS store to Azure Storage with Azure Data Box

8/22/2022 • 9 minutes to read • [Edit Online](#)

You can migrate data from an on-premises HDFS store of your Hadoop cluster into Azure Storage (blob storage or Data Lake Storage Gen2) by using a Data Box device. You can choose from Data Box Disk, an 80-TB Data Box or a 770-TB Data Box Heavy.

This article helps you complete these tasks:

- Prepare to migrate your data.
- Copy your data to a Data Box Disk, Data Box or a Data Box Heavy device.
- Ship the device back to Microsoft.
- Apply access permissions to files and directories (Data Lake Storage Gen2 only)

## Prerequisites

You need these things to complete the migration.

- An Azure Storage account.
- An on-premises Hadoop cluster that contains your source data.
- An [Azure Data Box device](#).
  - [Order your Data Box](#) or [Data Box Heavy](#).
  - Cable and connect your [Data Box](#) or [Data Box Heavy](#) to an on-premises network.

If you are ready, let's start.

## Copy your data to a Data Box device

If your data fits into a single Data Box device, then you'll copy the data to the Data Box device.

If your data size exceeds the capacity of the Data Box device, then use the [optional procedure to split the data across multiple Data Box devices](#) and then perform this step.

To copy the data from your on-premises HDFS store to a Data Box device, you'll set a few things up, and then use the [DistCp](#) tool.

Follow these steps to copy data via the REST APIs of Blob/Object storage to your Data Box device. The REST API interface will make the device appear as an HDFS store to your cluster.

1. Before you copy the data via REST, identify the security and connection primitives to connect to the REST interface on the Data Box or Data Box Heavy. Sign in to the local web UI of Data Box and go to **Connect and copy** page. Against the Azure storage account for your device, under **Access settings**, locate, and select REST.

The screenshot shows the Microsoft Azure Data Box web interface. In the top navigation bar, there are links for 'Lock device', 'Refresh', 'Help', 'Settings', and 'Sign out'. On the left, a sidebar menu includes 'View dashboard', 'Set network interfaces', 'Connect and copy' (which is highlighted with a red box), 'Prepare to ship', 'MANAGE' (with 'Shut down or restart', 'Contact Support', and 'Device reset'), and a 'NET' icon. The main content area is titled 'Connect and copy' and shows 'NetBackupPod01'. Below this, there's a table for 'AZURE STORAGE ACCOUNT' with one entry: 'mystorageaccount' in 'West US' with 0 objects and 0 errors. Under 'ACCESS SETTINGS', there are three tabs: 'SMB' (blue), 'NFS' (white), and 'REST' (red box). A red box also highlights the 'REST' tab.

2. In the Access storage account and upload data dialog, copy the **Blob service endpoint** and the **Storage account key**. From the blob service endpoint, omit the `https://` and the trailing slash.

In this case, the endpoint is: `https://mystorageaccount.blob.mydataboxno.microsoftdatabox.com/`. The host portion of the URI that you'll use is: `mystorageaccount.blob.mydataboxno.microsoftdatabox.com`. For an example, see how to [Connect to REST over http](#).

**Access storage account and upload data**

Use the storage account name and connection string to access the storage account and begin uploading data.

Storage Account Name  
📁 mystorageaccount

Key  
📁 myaccountkey

**Blob Service Endpoint**  
📁 `https://mystorageaccount.blob.mydataboxno.microsoftdatabox.com/`

Connection String  
📁 `DefaultEndpointsProtocol=https;EndpointSuffix= mydataboxno.microsoftdatabox.com;AccountName=mystorageaccount; AccountKey=myaccountkey`

**OK**

3. Add the endpoint and the Data Box or Data Box Heavy node IP address to `/etc/hosts` on each node.

```
10.128.5.42 mystorageaccount.blob.mydataboxno.microsoftdatabox.com
```

If you are using some other mechanism for DNS, you should ensure that the Data Box endpoint can be resolved.

4. Set the shell variable `azjars` to the location of the `hadoop-azure` and `azure-storage` jar files. You can find these files under the Hadoop installation directory.

To determine if these files exist, use the following command:

```
ls -l $<hadoop_install_dir>/share/hadoop/tools/lib/ | grep azure
```

Replace the `<hadoop_install_dir>` placeholder with the path to the directory where you've installed Hadoop. Be sure to use fully qualified paths.

Examples:

```
azjars=$hadoop_install_dir/share/hadoop/tools/lib/hadoop-azure-2.6.0-cdh5.14.0.jar
```

```
azjars=$azjars,$hadoop_install_dir/share/hadoop/tools/lib/microsoft-windowsstorage-sdk-0.6.0.jar
```

5. Create the storage container that you want to use for data copy. You should also specify a destination directory as part of this command. This could be a dummy destination directory at this point.

```
hadoop fs -libjars $azjars \
-D fs.AbstractFileSystem.wasb.impl=org.apache.hadoop.fs.azure.Wasb \
-D fs.azure.account.key.<blob_service_endpoint>=<account_key> \
mkdir -p wasb://<container_name>@<blob_service_endpoint>/<destination_directory>
```

- Replace the <blob\_service\_endpoint> placeholder with the name of your blob service endpoint.
- Replace the <account\_key> placeholder with the access key of your account.
- Replace the <container-name> placeholder with the name of your container.
- Replace the <destination\_directory> placeholder with the name of the directory that you want to copy your data to.

6. Run a list command to ensure that your container and directory were created.

```
hadoop fs -libjars $azjars \
-D fs.AbstractFileSystem.wasb.impl=org.apache.hadoop.fs.azure.Wasb \
-D fs.azure.account.key.<blob_service_endpoint>=<account_key> \
-ls -R wasb://<container_name>@<blob_service_endpoint>/
```

- Replace the <blob\_service\_endpoint> placeholder with the name of your blob service endpoint.
- Replace the <account\_key> placeholder with the access key of your account.
- Replace the <container-name> placeholder with the name of your container.

7. Copy data from the Hadoop HDFS to Data Box Blob storage, into the container that you created earlier. If the directory that you are copying into is not found, the command automatically creates it.

```
hadoop distcp \
-libjars $azjars \
-D fs.AbstractFileSystem.wasb.impl=org.apache.hadoop.fs.azure.Wasb \
-D fs.azure.account.key.<blob_service_endpoint>=<account_key> \
-filters <exclusion_filelist_file> \
[-f filelist_file | <source_directory> \
wasb://<container_name>@<blob_service_endpoint>/<destination_directory>
```

- Replace the <blob\_service\_endpoint> placeholder with the name of your blob service endpoint.
- Replace the <account\_key> placeholder with the access key of your account.
- Replace the <container-name> placeholder with the name of your container.
- Replace the <exclusion\_filelist\_file> placeholder with the name of the file that contains your list of file exclusions.
- Replace the <source\_directory> placeholder with the name of the directory that contains the data that you want to copy.
- Replace the <destination\_directory> placeholder with the name of the directory that you want to copy your data to.

The `-libjars` option is used to make the `hadoop-azure*.jar` and the dependent `azure-storage*.jar` files available to `distcp`. This may already occur for some clusters.

The following example shows how the `distcp` command is used to copy data.

```
hadoop distcp \
-libjars $azjars \
-D fs.AbstractFileSystem.wasb.impl=org.apache.hadoop.fs.azure.Wasb \
-D fs.azure.account.key.mystorageaccount.blob.mydataboxno.microsoftdatabox.com=myaccountkey \
-filter ./exclusions.lst -f /tmp/copylist1 -m 4 \
/data/testfiles \
wasb://hdfscontainer@mystorageaccount.blob.mydataboxno.microsoftdatabox.com/data
```

To improve the copy speed:

- Try changing the number of mappers. (The default number of mappers is 20. The above example uses `m = 4` mappers.)
- Try `-D fs.azure.concurrentRequestCount.out=<thread_number>`. Replace `<thread_number>` with the number of threads per mapper. The product of the number of mappers and the number of threads per mapper, `m*<thread_number>`, should not exceed 32.
- Try running multiple `distcp` in parallel.
- Remember that large files perform better than small files.
- If you have files larger than 200 GB, we recommend changing the block size to 100MB with the following parameters:

```
hadoop distcp \
-libjars $azjars \
-Dfs.azure.write.request.size= 104857600 \
-Dfs.AbstractFileSystem.wasb.impl=org.apache.hadoop.fs.azure.Wasb \
-Dfs.azure.account.key.<blob_service_endpoint>=<account_key> \
-strategy dynamic \
-Dmapreduce.map.memory.mb=16384 \
-Dfs.azure.concurrentRequestCount.out=8 \
-Dmapreduce.map.java.opts=-Xmx8196m \
-m 4 \
-update \
/data/bigfile wasb://hadoop@mystorageaccount.blob.core.windows.net/bigfile
```

## Ship the Data Box to Microsoft

Follow these steps to prepare and ship the Data Box device to Microsoft.

1. First, [Prepare to ship on your Data Box or Data Box Heavy](#).
2. After the device preparation is complete, download the BOM files. You will use these BOM or manifest files later to verify the data uploaded to Azure.
3. Shut down the device and remove the cables.
4. Schedule a pickup with UPS.
  - For Data Box devices, see [Ship your Data Box](#).
  - For Data Box Heavy devices, see [Ship your Data Box Heavy](#).
5. After Microsoft receives your device, it is connected to the data center network and the data is uploaded to the storage account you specified when you placed the device order. Verify against the BOM files that all your data is uploaded to Azure.

# Apply access permissions to files and directories (Data Lake Storage Gen2 only)

You already have the data into your Azure Storage account. Now you will apply access permissions to files and directories.

## NOTE

This step is needed only if you are using Azure Data Lake Storage Gen2 as your data store. If you are using just a blob storage account without hierarchical namespace as your data store, you can skip this section.

## Create a service principal for your Azure Data Lake Storage Gen2 account

To create a service principal, see [How to: Use the portal to create an Azure AD application and service principal that can access resources](#).

- When performing the steps in the [Assign the application to a role](#) section of the article, make sure to assign the **Storage Blob Data Contributor** role to the service principal.
- When performing the steps in the [Get values for signing in](#) section of the article, save application ID, and client secret values into a text file. You'll need those soon.

## Generate a list of copied files with their permissions

From the on-premises Hadoop cluster, run this command:

```
sudo -u hdfs ./copy-acls.sh -s /{hdfs_path} > ./filelist.json
```

This command generates a list of copied files with their permissions.

## NOTE

Depending on the number of files in the HDFS, this command can take a long time to run.

## Generate a list of identities and map them to Azure Active Directory (ADD) identities

1. Download the `copy-acls.py` script. See the [Download helper scripts and set up your edge node to run them](#) section of this article.
2. Run this command to generate a list of unique identities.

```
./copy-acls.py -s ./filelist.json -i ./id_map.json -g
```

This script generates a file named `id_map.json` that contains the identities that you need to map to ADD-based identities.

3. Open the `id_map.json` file in a text editor.
4. For each JSON object that appears in the file, update the `target` attribute of either an AAD User Principal Name (UPN) or ObjectId (OID), with the appropriate mapped identity. After you're done, save the file. You'll need this file in the next step.

## Apply permissions to copied files and apply identity mappings

Run this command to apply permissions to the data that you copied into the Data Lake Storage Gen2 account:

```
./copy-acls.py -s ./filelist.json -i ./id_map.json -A <storage-account-name> -C <container-name> --dest-spn-id <application-id> --dest-spn-secret <client-secret>
```

- Replace the `<storage-account-name>` placeholder with the name of your storage account.
- Replace the `<container-name>` placeholder with the name of your container.
- Replace the `<application-id>` and `<client-secret>` placeholders with the application ID and client secret that you collected when you created the service principal.

## Appendix: Split data across multiple Data Box devices

Before you move your data onto a Data Box device, you'll need to download some helper scripts, ensure that your data is organized to fit onto a Data Box device, and exclude any unnecessary files.

### Download helper scripts and set up your edge node to run them

1. From your edge or head node of your on-premises Hadoop cluster, run this command:

```
git clone https://github.com/jamesbak/databox-adls-loader.git
cd databox-adls-loader
```

This command clones the GitHub repository that contains the helper scripts.

2. Make sure that have the `jq` package installed on your local computer.

```
sudo apt-get install jq
```

3. Install the [Requests](#) python package.

```
pip install requests
```

4. Set execute permissions on the required scripts.

```
chmod +x *.py *.sh
```

### Ensure that your data is organized to fit onto a Data Box device

If the size of your data exceeds the size of a single Data Box device, you can split files up into groups that you can store onto multiple Data Box devices.

If your data doesn't exceed the size of a singe Data Box device, you can proceed to the next section.

1. With elevated permissions, run the `generate-file-list` script that you downloaded by following the guidance in the previous section.

Here's a description of the command parameters:

```

sudo -u hdfs ./generate-file-list.py [-h] [-s DATABOX_SIZE] [-b FILELIST_BASENAME]
[-f LOG_CONFIG] [-l LOG_FILE]
[-v {DEBUG,INFO,WARNING,ERROR}]
path

where:
positional arguments:
path The base HDFS path to process.

optional arguments:
-h, --help show this help message and exit
-s DATABOX_SIZE, --databox-size DATABOX_SIZE
 The size of each Data Box in bytes.
-b FILELIST_BASENAME, --filelist-basename FILELIST_BASENAME
 The base name for the output filelists. Lists will be
 named basename1, basename2,
-f LOG_CONFIG, --log-config LOG_CONFIG
 The name of a configuration file for logging.
-l LOG_FILE, --log-file LOG_FILE
 Name of file to have log output written to (default is
 stdout/stderr)
-v {DEBUG,INFO,WARNING,ERROR}, --log-level {DEBUG,INFO,WARNING,ERROR}
 Level of log information to output. Default is 'INFO'.

```

2. Copy the generated file lists to HDFS so that they are accessible to the [DistCp](#) job.

```

hadoop fs -copyFromLocal {filelist_pattern} /[hdfs directory]

```

### Exclude unnecessary files

You'll need to exclude some directories from the DisCp job. For example, exclude directories that contain state information that keep the cluster running.

On the on-premises Hadoop cluster where you plan to initiate the DistCp job, create a file that specifies the list of directories that you want to exclude.

Here's an example:

```

.*ranger/audit.*
.*/hbase/data/WALs.*

```

## Next steps

Learn how Data Lake Storage Gen2 works with HDInsight clusters. For more information, see [Use Azure Data Lake Storage Gen2 with Azure HDInsight clusters](#).

# Migrate on-premises Hadoop data to Azure Data Lake Storage Gen2 with WANdisco LiveData Platform for Azure

8/22/2022 • 5 minutes to read • [Edit Online](#)

[WANdisco LiveData Platform for Azure](#) migrates petabytes of on-premises Hadoop data to Azure Data Lake Storage Gen2 file systems without interrupting data operations or requiring downtime. The platform's continuous checks prevent data from being lost while keeping it consistent at both ends of transference even while it undergoes modification.

The platform consists of two services. [LiveData Migrator for Azure](#) migrates actively used data from on-premises environments to Azure storage, and [LiveData Plane for Azure](#) ensures that all modified or ingested data is replicated consistently.



Manage both services by using the Azure portal and the Azure CLI. Each service follows the same metered, pay-as-you-go billing model as all other Azure services: data consumption in LiveData Platform for Azure will appear on the monthly Azure bill, which will provide usage metrics.

Unlike migrating data *offline* by [copying static information to Azure Data Box](#), or by using Hadoop tools like [DistCp](#), you can maintain full operation of your business systems during *online* migration with WANdisco LiveData for Azure. Keep your big data environments operating even while moving their data to Azure.

## Key benefits of WANdisco LiveData Platform for Azure

[WANdisco LiveData Platform for Azure](#)'s wide-area network capable consensus engine achieves data consistency, and conducts real-time data replication at scale. See the following video for more information:

Key benefits of the platform include the following:

- **Data accuracy:** End-to-end validation of data prevents data loss and ensures transferred data is fit for

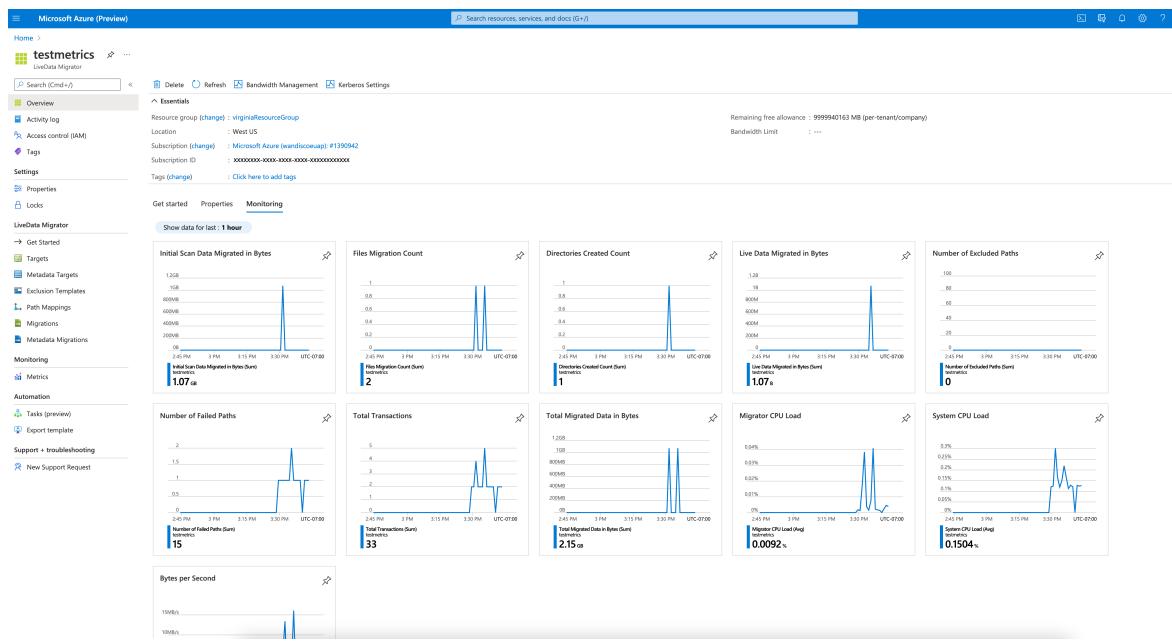
use.

- **Data consistency:** Keep data volumes automatically consistent between environments even while they undergo continuous change.
- **Data efficiency:** Transfer large data volumes continuously with full control of bandwidth consumption.
- **Downtime elimination:** Freely create, modify, read, and delete data with other applications during migration, without the need to disrupt business operations during data transference to Azure. Continue to operate applications, analytics infrastructure, ingest jobs, and other processing.
- **Simple use:** Use the Platform's Azure integration to create, configure, schedule, and track the progress of automated migrations. Additionally, configure selective data replication, Hive metadata, data security, and confidentiality as needed.

## Key features of WANdisco LiveData Platform for Azure

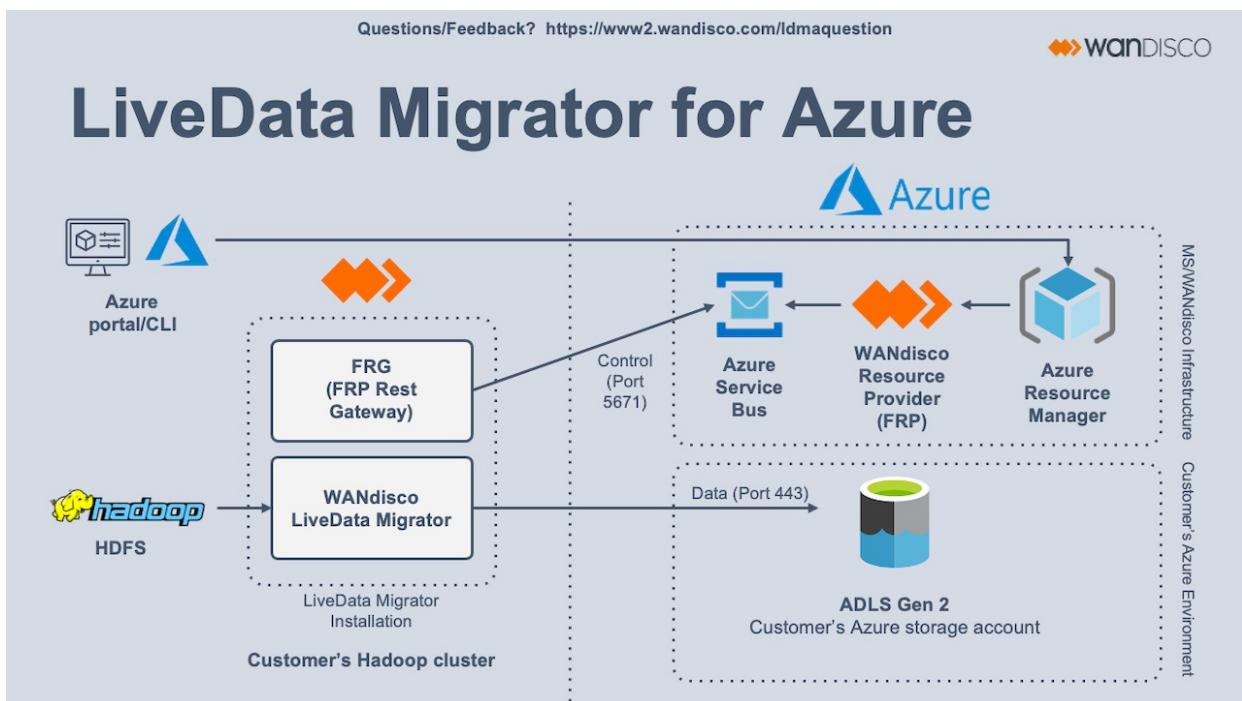
Key features of the platform include the following:

- **Metadata Migration:** In addition to HDFS data, migrate metadata (from Hive and other storages) with LiveData Migrator for Azure.
- **Scheduled Transfer:** Use LiveData Migrator for Azure to control and automate when data transfer will initiate, eliminating the need to manually migrate changes to data.
- **Kerberos:** LiveData Migrator for Azure supports Kerberized clusters.
- **Exclusion Templates:** Create rules in LiveData Migrator for Azure to prevent certain file sizes or file names (defined using glob patterns) from being migrated to your target storage. Create exclusion templates in the Azure portal or with the CLI, and apply them to any number of migrations.
- **Path Mappings:** Define alternate target paths for specific target file systems, which automatically move transferred data to directories you specify.
- **Bandwidth Management:** Configure the maximum amount of network bandwidth LiveData Migrator for Azure can use to prevent bandwidth over consumption.
- **Exclusions:** Define template queries that prevent the migration of any files and directories that meet the criteria, allowing you to selectively migrate data from your source system.
- **Metrics:** View details about data transfer in LiveData Migrator for Azure, such as files transferred over time, excluded paths, items that failed to transfer and more.



## Migrate big data faster without risk

The first service included in WANdisco LiveData Platform for Azure is [LiveData Migrator for Azure](#), which migrates data from on-premises environments to Azure Storage. Once you've deployed LiveData Migrator to your on-premises Hadoop cluster, it will automatically create the best configuration for your file system. From there, supply the Kerberos details for the system. LiveData Migrator for Azure will then be ready to migrate data to Azure Storage.



Before you start with LiveData Migrator for Azure, review these [prerequisites](#).

To perform a migration:

1. In the Azure CLI:

- Register for the WANdisco resource provider in the Azure CLI by running

```
az provider register --namespace Wandisco.Fusion --consent-to-permissions .
```

- Accept the metered billing terms of LiveData Platform by running

```
az vm image terms accept --offer ldma --plan metered-v1 --publisher Wandisco --subscription <subscriptionID>
```

2. Deploy a LiveData Migrator instance from the Azure portal to your on-premises Hadoop cluster. (You do not need to make changes to or restart the cluster.)

Microsoft Azure

Search resources, services, and docs (G+)

Home > LiveData Migrator for Azure >

## Create LiveData Migrator for Azure

Basics Tags Review + create

**Project details**

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

Subscription \* ⓘ Microsoft Azure (wandiscoinc) #1284731

Resource group \* ⓘ SUPPORT-joe.siolk1

Create new

**Instance details**

Region \* ⓘ West Central US

LiveData Migrator Name \* ⓘ testmigrator

Do you need a Hadoop test cluster? \* ⓘ  Yes  No

Cluster Admin Username \* ⓘ clusteradmin

Password \* ⓘ ..... Confirm password \* ⓘ .....

Review + create < Previous Next : Tags >

**NOTE**

WANDisco LiveData Migrator for Azure provides the option to create a Hadoop Test Cluster.

3. Configure Kerberos details, if applicable.
4. Define the target Azure Data Lake Storage Gen2-enabled storage account.

[Home](#) > [wandisco.ldm-preview-20201005155212](#) > [simple1](#) >

## Create Target

X
[Basics](#)   [Tags](#)   [Review + create](#)

Please enter the name of the Target you want to deploy

Target Name \* ⓘ



### Storage details

Storage Account \* ⓘ



Storage Container \* ⓘ



Filter items...

### Project details

Resource subscription:

Resource Group: psm-thursday-sessions

Resource location: westus

Migrator: simple1

[Review + create](#)
[< Previous](#)
[Next : Tags >](#)

5. Define the location of the data that you want to migrate, for example: `/user/hive/warehouse`.

[Home](#) > [SUPPORT-joe.sio1](#) > [testmigrator](#) >

## Create Migration

...
[Basics](#)   [Migration Settings](#)   [Tags](#)   [Review + create](#)

Migration Name \* ⓘ



Target Storage \* ⓘ



Path \* ⓘ



### Project details

Resource subscription: 3842fefa-7697-4e7d-b051-a5a3ae601030

Resource Group: SUPPORT-joe.sio1

Resource location: westcentralus

LiveData Migrator: testmigrator

[Review + create](#)
[< Previous](#)
[Next : Migration Settings >](#)

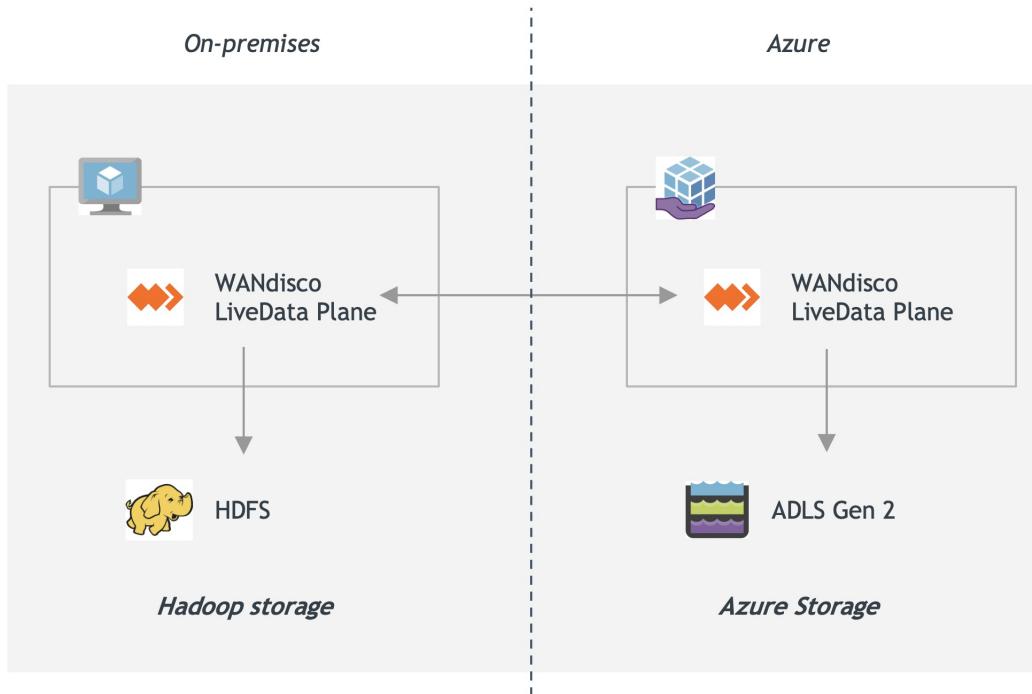
6. Start the migration.

Monitor your migration progress through standard Azure tooling including the Azure CLI and Azure portal.

For more detailed instructions, see the [LiveData Migrator for Azure How-To video series](#).

# Bidirectionally replicate data under active change with LiveData Plane for Azure

The second service included in the LiveData Platform is [LiveData Plane for Azure](#). LiveData Plane uses WANdisco's coordination engine to keep data consistent across many on-premises Hadoop clusters and Azure Storage by intelligently applying changes to data on all systems, removing the risk of data conflicts at different points of use.



After initial migration, keep your data consistent with LiveData Plane for Azure:

1. Deploy LiveData Plane for Azure on-premises and in Azure, starting from the Azure portal. No application changes are required.
2. Configure replication rules that cover the data locations that you want to keep consistent, for example:  
`/user/contoso/sales/region/WA`.
3. Run applications that access and modify data in either location as you need.

LiveData Plane for Azure consistently replicates data changes across all environments without significant impact on cluster operation or application performance.

## Test drive or Trial

From [LiveData Platform for Azure's Marketplace page](#), you have two options:

- The **Get It Now** button launches the service in your subscription. From there, you may use your own Hadoop cluster or WANdisco's Trial cluster.
- Click **Test Drive** to test LiveData Migrator for Azure in an environment that is preconfigured and hosted for you. This enables you to try LiveData Migrator for Azure before adding it to your subscription, without any cost or risk to your data.

Watch the [Test Drive Demonstration Video](#) to see the test drive in action.

## Next Steps

- [Plan and create a migration in LiveData Migrator for Azure.](#)

## See also

- [LiveData Migrator for Azure on Azure Marketplace](#)
- [LiveData Migrator for Azure plans and pricing](#)
- [LiveData Platform for Azure Frequently Asked Questions](#)
- [Known Issues with LiveData Platform for Azure](#)

# Get started with AzCopy

8/22/2022 • 6 minutes to read • [Edit Online](#)

AzCopy is a command-line utility that you can use to copy blobs or files to or from a storage account. This article helps you download AzCopy, connect to your storage account, and then transfer data.

## NOTE

AzCopy **V10** is the currently supported version of AzCopy.

If you need to use a previous version of AzCopy, see the [Use the previous version of AzCopy](#) section of this article.

## Download AzCopy

First, download the AzCopy V10 executable file to any directory on your computer. AzCopy V10 is just an executable file, so there's nothing to install.

- [Windows 64-bit \(zip\)](#)
- [Windows 32-bit \(zip\)](#)
- [Linux x86-64 \(tar\)](#)
- [Linux ARM64 Preview \(tar\)](#)
- [macOS \(zip\)](#)

These files are compressed as a zip file (Windows and Mac) or a tar file (Linux). To download and decompress the tar file on Linux, see the documentation for your Linux distribution.

For detailed information on AzCopy releases see the [AzCopy release page](#).

## NOTE

If you want to copy data to and from your [Azure Table storage](#) service, then install [AzCopy version 7.3](#).

## Run AzCopy

For convenience, consider adding the directory location of the AzCopy executable to your system path for ease of use. That way you can type `azcopy` from any directory on your system.

If you choose not to add the AzCopy directory to your path, you'll have to change directories to the location of your AzCopy executable and type `azcopy` or `.\azcopy` in Windows PowerShell command prompts.

As an owner of your Azure Storage account, you aren't automatically assigned permissions to access data. Before you can do anything meaningful with AzCopy, you need to decide how you'll provide authorization credentials to the storage service.

## Authorize AzCopy

You can provide authorization credentials by using Azure Active Directory (AD), or by using a Shared Access Signature (SAS) token.

Use this table as a guide:

STORAGE TYPE	CURRENTLY SUPPORTED METHOD OF AUTHORIZATION
<b>Blob storage</b>	Azure AD & SAS
<b>Blob storage (hierarchical namespace)</b>	Azure AD & SAS
<b>File storage</b>	SAS only

#### Option 1: Use Azure Active Directory

This option is available for blob Storage only. By using Azure Active Directory, you can provide credentials once instead of having to append a SAS token to each command.

#### Option 2: Use a SAS token

You can append a SAS token to each source or destination URL that use in your AzCopy commands.

This example command recursively copies data from a local directory to a blob container. A fictitious SAS token is appended to the end of the container URL.

```
azcopy copy "C:\local\path" "https://account.blob.core.windows.net/mycontainer1/?sv=2018-03-28&ss=bjqt&srt=sco&sp=rwddgcup&se=2019-05-01T05:01:17Z&st=2019-04-30T21:01:17Z&spr=https&sig=MGCXiyezbttkr3ewJih2AR8Krghsy1DGM9ovN734bQF4%3D" --recursive=true
```

To learn more about SAS tokens and how to obtain one, see [Using shared access signatures \(SAS\)](#).

#### NOTE

The [Secure transfer required](#) setting of a storage account determines whether the connection to a storage account is secured with Transport Layer Security (TLS). This setting is enabled by default.

## Transfer data

After you've authorized your identity or obtained a SAS token, you can begin transferring data.

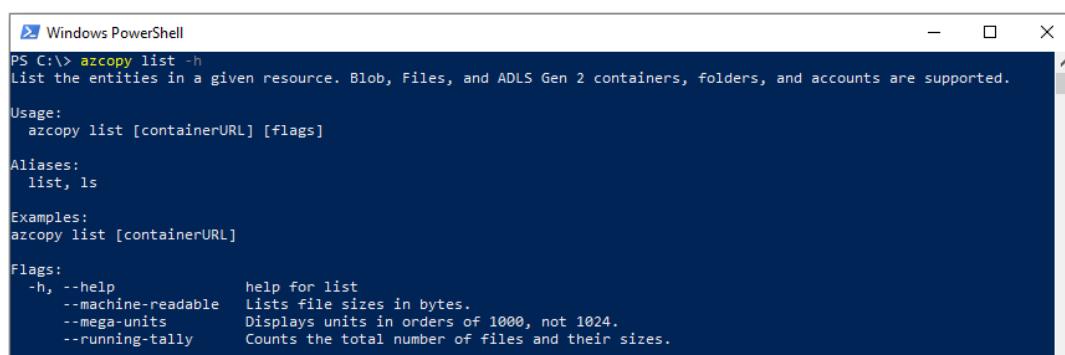
To find example commands, see any of these articles.

SERVICE	ARTICLE
Azure Blob Storage	<a href="#">Upload files to Azure Blob Storage</a>
Azure Blob Storage	<a href="#">Download blobs from Azure Blob Storage</a>
Azure Blob Storage	<a href="#">Copy blobs between Azure storage accounts</a>
Azure Blob Storage	<a href="#">Synchronize with Azure Blob Storage</a>
Azure Files	<a href="#">Transfer data with AzCopy and file storage</a>
Amazon S3	<a href="#">Copy data from Amazon S3 to Azure Storage</a>
Google Cloud Storage	<a href="#">Copy data from Google Cloud Storage to Azure Storage (preview)</a>
Azure Stack storage	<a href="#">Transfer data with AzCopy and Azure Stack storage</a>

## Get command help

To see a list of commands, type `azcopy -h` and then press the ENTER key.

To learn about a specific command, just include the name of the command (For example: `azcopy list -h`).



```
PS C:\> azcopy list -h
List the entities in a given resource. Blob, Files, and ADLS Gen 2 containers, folders, and accounts are supported.

Usage:
 azcopy list [containerURL] [flags]

Aliases:
 list, ls

Examples:
azcopy list [containerURL]

Flags:
 -h, --help help for list
 --machine-readable Lists file sizes in bytes.
 --mega-units Displays units in orders of 1000, not 1024.
 --running-tally Counts the total number of files and their sizes.
```

## List of commands

The following table lists all AzCopy v10 commands. Each command links to a reference article.

COMMAND	DESCRIPTION
<a href="#">azcopy bench</a>	Runs a performance benchmark by uploading or downloading test data to or from a specified location.
<a href="#">azcopy copy</a>	Copies source data to a destination location
<a href="#">azcopy doc</a>	Generates documentation for the tool in Markdown format.
<a href="#">azcopy env</a>	Shows the environment variables that can configure AzCopy's behavior.
<a href="#">azcopy jobs</a>	Subcommands related to managing jobs.
<a href="#">azcopy jobs clean</a>	Remove all log and plan files for all jobs.
<a href="#">azcopy jobs list</a>	Displays information on all jobs.
<a href="#">azcopy jobs remove</a>	Remove all files associated with the given job ID.
<a href="#">azcopy jobs resume</a>	Resumes the existing job with the given job ID.
<a href="#">azcopy jobs show</a>	Shows detailed information for the given job ID.
<a href="#">azcopy list</a>	Lists the entities in a given resource.
<a href="#">azcopy login</a>	Logs in to Azure Active Directory to access Azure Storage resources.
<a href="#">azcopy login status</a>	Lists the entities in a given resource.
<a href="#">azcopy logout</a>	Logs the user out and terminates access to Azure Storage resources.
<a href="#">azcopy make</a>	Creates a container or file share.
<a href="#">azcopy remove</a>	Delete blobs or files from an Azure storage account.
<a href="#">azcopy sync</a>	Replicates the source location to the destination location.

### NOTE

AzCopy does not have a command to rename files.

## Use in a script

### Obtain a static download link

Over time, the AzCopy [download link](#) will point to new versions of AzCopy. If your script downloads AzCopy, the script might stop working if a newer version of AzCopy modifies features that your script depends upon.

To avoid these issues, obtain a static (unchanging) link to the current version of AzCopy. That way, your script downloads the same exact version of AzCopy each time that it runs.

To obtain the link, run this command:

OPERATING SYSTEM	COMMAND
Linux	<pre>curl -s -D- https://aka.ms/downloadazcopy-v10-linux   grep ^Location</pre>

OPERATING SYSTEM	COMMAND
Windows (PowerShell Core 7)	(Invoke-WebRequest https://aka.ms/downloadazcopy-v10-windows -MaximumRedirection 0 -ErrorAction silentlycontinue -SkipHttpErrorCheck).headers.location[0]
Windows (PowerShell 5.1)	(Invoke-WebRequest https://aka.ms/downloadazcopy-v10-windows -MaximumRedirection 0 -ErrorAction silentlycontinue ).headers.location

#### NOTE

For Linux, `--strip-components=1` on the `tar` command removes the top-level folder that contains the version name, and instead extracts the binary directly into the current folder. This allows the script to be updated with a new version of `azcopy` by only updating the `wget` URL.

The URL appears in the output of this command. Your script can then download AzCopy by using that URL.

OPERATING SYSTEM	COMMAND
Linux	wget -O azcopy_v10.tar.gz https://aka.ms/downloadazcopy-v10-linux && tar -xvf azcopy_v10.tar.gz --strip-components=1
Windows	Invoke-WebRequest https://azcopyvnext.azureedge.net/release20190517/azcopy_windows_amd64_1 -Outfile azcopyv10.zip <<Unzip here>>

#### Escape special characters in SAS tokens

In batch files that have the `.cmd` extension, you'll have to escape the `%` characters that appear in SAS tokens. You can do that by adding an additional `%` character next to existing `%` characters in the SAS token string.

#### Run scripts by using Jenkins

If you plan to use [Jenkins](#) to run scripts, make sure to place the following command at the beginning of the script.

```
/usr/bin/keyctl new_session
```

## Use in Azure Storage Explorer

[Storage Explorer](#) uses AzCopy to perform all of its data transfer operations. You can use [Storage Explorer](#) if you want to leverage the performance advantages of AzCopy, but you prefer to use a graphical user interface rather than the command line to interact with your files.

Storage Explorer uses your account key to perform operations, so after you sign into Storage Explorer, you won't need to provide additional authorization credentials.

## Configure, optimize, and fix

See any of the following resources:

- [AzCopy configuration settings](#)
- [Optimize the performance of AzCopy](#)
- [Find errors and resume jobs by using log and plan files in AzCopy](#)
- [Troubleshoot problems with AzCopy v10](#)

## Use a previous version

If you need to use the previous version of AzCopy, see either of the following links:

- [AzCopy on Windows \(v8\)](#)

- [AzCopy on Linux \(v7\)](#)

## Next steps

If you have questions, issues, or general feedback, submit them [on GitHub](#) page.

# Authorize access to blobs with AzCopy and Azure Active Directory (Azure AD)

8/22/2022 • 10 minutes to read • [Edit Online](#)

You can provide AzCopy with authorization credentials by using Azure AD. That way, you won't have to append a shared access signature (SAS) token to each command.

Start by verifying your role assignments. Then, choose what type of *security principal* you want to authorize. A [user identity](#), a [managed identity](#), and a [service principal](#) are each a type of security principal.

A user identity is any user that has an identity in Azure AD. It's the easiest security principal to authorize. Managed identities and service principals are great options if you plan to use AzCopy inside of a script that runs without user interaction. A managed identity is better suited for scripts that run from an Azure Virtual Machine (VM), and a service principal is better suited for scripts that run on-premises.

For more information about AzCopy, [Get started with AzCopy](#).

## Verify role assignments

The level of authorization that you need is based on whether you plan to upload files or just download them.

If you just want to download files, then verify that the [Storage Blob Data Reader](#) role has been assigned to your user identity, managed identity, or service principal.

If you want to upload files, then verify that one of these roles has been assigned to your security principal:

- [Storage Blob Data Contributor](#)
- [Storage Blob Data Owner](#)

These roles can be assigned to your security principal in any of these scopes:

- Container (file system)
- Storage account
- Resource group
- Subscription

To learn how to verify and assign roles, see [Assign an Azure role for access to blob data](#).

### NOTE

Keep in mind that Azure role assignments can take up to five minutes to propagate.

You don't need to have one of these roles assigned to your security principal if your security principal is added to the access control list (ACL) of the target container or directory. In the ACL, your security principal needs write permission on the target directory, and execute permission on container and each parent directory.

To learn more, see [Access control model in Azure Data Lake Storage Gen2](#).

## Authorize a user identity

After you've verified that your user identity has been given the necessary authorization level, open a command prompt, type the following command, and then press the ENTER key.

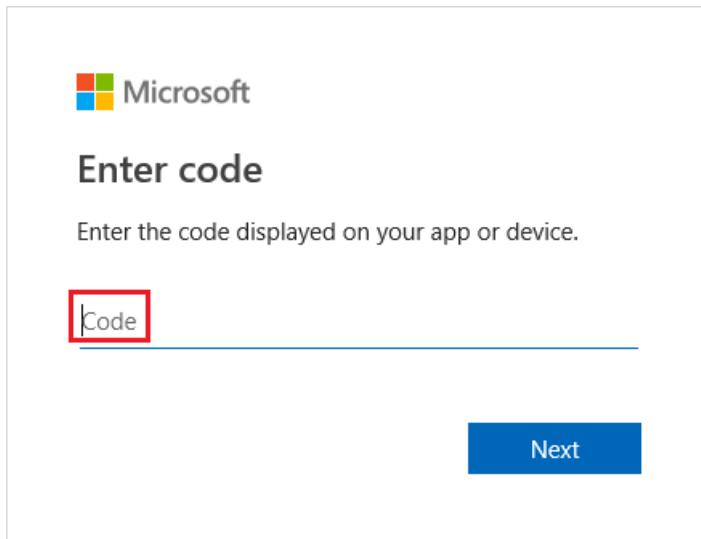
```
azcopy login
```

If you receive an error, try including the tenant ID of the organization to which the storage account belongs.

```
azcopy login --tenant-id=<tenant-id>
```

Replace the `<tenant-id>` placeholder with the tenant ID of the organization to which the storage account belongs. To find the tenant ID, select **Azure Active Directory > Properties > Directory ID** in the Azure portal.

This command returns an authentication code and the URL of a website. Open the website, provide the code, and then choose the **Next** button.



A sign-in window will appear. In that window, sign into your Azure account by using your Azure account credentials. After you've successfully signed in, you can close the browser window and begin using AzCopy.

## Authorize a managed identity

This is a great option if you plan to use AzCopy inside of a script that runs without user interaction, and the script runs from an Azure Virtual Machine (VM). When using this option, you won't have to store any credentials on the VM.

You can sign into your account by using a system-wide managed identity that you've enabled on your VM, or by using the client ID, Object ID, or Resource ID of a user-assigned managed identity that you've assigned to your VM.

To learn more about how to enable a system-wide managed identity or create a user-assigned managed identity, see [Configure managed identities for Azure resources on a VM using the Azure portal](#).

### Authorize by using a system-wide managed identity

First, make sure that you've enabled a system-wide managed identity on your VM. See [System-assigned managed identity](#).

Then, in your command console, type the following command, and then press the ENTER key.

```
azcopy login --identity
```

### Authorize by using a user-assigned managed identity

First, make sure that you've enabled a user-assigned managed identity on your VM. See [User-assigned managed](#)

identity.

Then, in your command console, type any of the following commands, and then press the ENTER key.

```
azcopy login --identity --identity-client-id "<client-id>"
```

Replace the `<client-id>` placeholder with the client ID of the user-assigned managed identity.

```
azcopy login --identity --identity-object-id "<object-id>"
```

Replace the `<object-id>` placeholder with the object ID of the user-assigned managed identity.

```
azcopy login --identity --identity-resource-id "<resource-id>"
```

Replace the `<resource-id>` placeholder with the resource ID of the user-assigned managed identity.

## Authorize a service principal

This is a great option if you plan to use AzCopy inside of a script that runs without user interaction, particularly when running on-premises. If you plan to run AzCopy on VMs that run in Azure, a managed service identity is easier to administer. To learn more, see the [Authorize a managed identity](#) section of this article.

Before you run a script, you have to sign in interactively at least one time so that you can provide AzCopy with the credentials of your service principal. Those credentials are stored in a secured and encrypted file so that your script doesn't have to provide that sensitive information.

You can sign into your account by using a client secret or by using the password of a certificate that is associated with your service principal's app registration.

To learn more about creating service principal, see [How to: Use the portal to create an Azure AD application and service principal that can access resources](#).

To learn more about service principals in general, see [Application and service principal objects in Azure Active Directory](#)

### Authorize a service principal by using a client secret

Start by setting the `AZCOPY_SPA_CLIENT_SECRET` environment variable to the client secret of your service principal's app registration.

#### NOTE

Make sure to set this value from your command prompt, and not in the environment variable settings of your operating system. That way, the value is available only to the current session.

This example shows how you could do this in PowerShell.

```
$env:AZCOPY_SPA_CLIENT_SECRET="$(Read-Host -prompt "Enter key")"
```

#### NOTE

Consider using a prompt as shown in this example. That way, your password won't appear in your console's command history.

Next, type the following command, and then press the ENTER key.

```
azcopy login --service-principal --application-id application-id --tenant-id=tenant-id
```

Replace the `<application-id>` placeholder with the application ID of your service principal's app registration. Replace the `<tenant-id>` placeholder with the tenant ID of the organization to which the storage account belongs. To find the tenant ID, select **Azure Active Directory > Properties > Directory ID** in the Azure portal.

#### Authorize a service principal by using a certificate

If you prefer to use your own credentials for authorization, you can upload a certificate to your app registration, and then use that certificate to log in.

In addition to uploading your certificate to your app registration, you'll also need to have a copy of the certificate saved to the machine or VM where AzCopy will be running. This copy of the certificate should be in .PFX or .PEM format, and must include the private key. The private key should be password-protected. If you're using Windows, and your certificate exists only in a certificate store, make sure to export that certificate to a PFX file (including the private key). For guidance, see [Export-PfxCertificate](#)

Next, set the `AZCOPY_SPA_CERT_PASSWORD` environment variable to the certificate password.

#### NOTE

Make sure to set this value from your command prompt, and not in the environment variable settings of your operating system. That way, the value is available only to the current session.

This example shows how you could do this task in PowerShell.

```
$env:AZCOPY_SPA_CERT_PASSWORD="$(Read-Host -prompt "Enter key")"
```

Next, type the following command, and then press the ENTER key.

```
azcopy login --service-principal --certificate-path <path-to-certificate-file> --tenant-id=<tenant-id>
```

Replace the `<path-to-certificate-file>` placeholder with the relative or fully qualified path to the certificate file. AzCopy saves the path to this certificate but it doesn't save a copy of the certificate, so make sure to keep that certificate in place. Replace the `<tenant-id>` placeholder with the tenant ID of the organization to which the storage account belongs. To find the tenant ID, select **Azure Active Directory > Properties > Directory ID** in the Azure portal.

#### NOTE

Consider using a prompt as shown in this example. That way, your password won't appear in your console's command history.

## Authorize without a secret store

The `azcopy login` command retrieves an OAuth token and then places that token into a secret store on your system. If your operating system doesn't have a secret store such as a Linux *keyring*, the `azcopy login` command won't work because there is nowhere to place the token.

Instead of using the `azcopy login` command, you can set in-memory environment variables. Then run any

AzCopy command. AzCopy will retrieve the Auth token required to complete the operation. After the operation completes, the token disappears from memory.

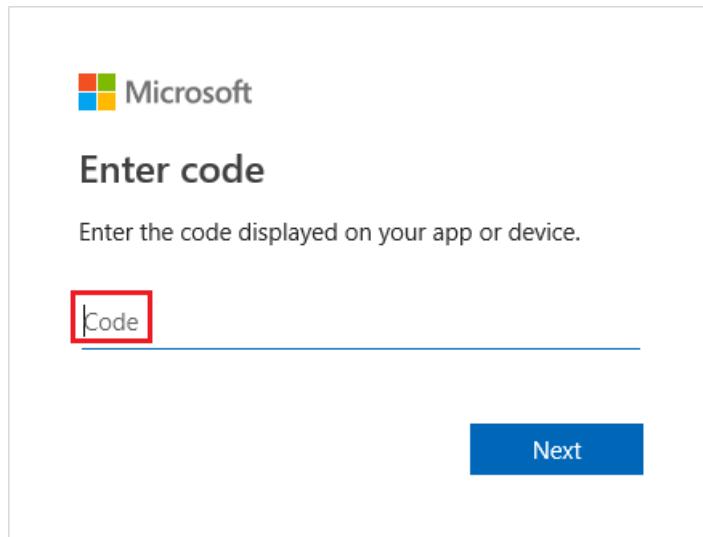
### Authorize a user identity

After you've verified that your user identity has been given the necessary authorization level, type the following command, and then press the ENTER key.

```
export AZCOPY_AUTO_LOGIN_TYPE=DEVICE
```

Then, run any azcopy command (For example: `azcopy list https://contoso.blob.core.windows.net`).

This command returns an authentication code and the URL of a website. Open the website, provide the code, and then choose the **Next** button.



A sign-in window will appear. In that window, sign into your Azure account by using your Azure account credentials. After you've successfully signed in, the operation can complete.

### Authorize by using a system-wide managed identity

First, make sure that you've enabled a system-wide managed identity on your VM. See [System-assigned managed identity](#).

Type the following command, and then press the ENTER key.

```
export AZCOPY_AUTO_LOGIN_TYPE=MSI
```

Then, run any azcopy command (For example: `azcopy list https://contoso.blob.core.windows.net`).

### Authorize by using a user-assigned managed identity

First, make sure that you've enabled a user-assigned managed identity on your VM. See [User-assigned managed identity](#).

Type the following command, and then press the ENTER key.

```
export AZCOPY_AUTO_LOGIN_TYPE=MSI
```

Then, type any of the following commands, and then press the ENTER key.

```
export AZCOPY_MSI_CLIENT_ID=<client-id>
```

Replace the `<client-id>` placeholder with the client ID of the user-assigned managed identity.

```
export AZCOPY_MSI_OBJECT_ID=<object-id>
```

Replace the `<object-id>` placeholder with the object ID of the user-assigned managed identity.

```
export AZCOPY_MSI_RESOURCE_STRING=<resource-id>
```

Replace the `<resource-id>` placeholder with the resource ID of the user-assigned managed identity.

After you set these variables, you can run any azcopy command (For example:

```
azcopy list https://contoso.blob.core.windows.net).
```

## Authorize a service principal

You can sign into your account by using a client secret or by using the password of a certificate that is associated with your service principal's app registration.

### Authorize a service principal by using a client secret

Type the following command, and then press the ENTER key.

```
export AZCOPY_AUTO_LOGIN_TYPE=SPN
export AZCOPY_SPA_APPLICATION_ID=<application-id>
export AZCOPY_SPA_CLIENT_SECRET=<client-secret>
export AZCOPY_TENANT_ID=<tenant-id>
```

Replace the `<application-id>` placeholder with the application ID of your service principal's app registration.

Replace the `<client-secret>` placeholder with the client secret. Replace the `<tenant-id>` placeholder with the tenant ID of the organization to which the storage account belongs. To find the tenant ID, select **Azure Active Directory > Properties > Directory ID** in the Azure portal.

#### NOTE

Consider using a prompt to collect the password from the user. That way, your password won't appear in your command history.

Then, run any azcopy command (For example: `azcopy list https://contoso.blob.core.windows.net`).

### Authorize a service principal by using a certificate

If you prefer to use your own credentials for authorization, you can upload a certificate to your app registration, and then use that certificate to log in.

In addition to uploading your certificate to your app registration, you'll also need to have a copy of the certificate saved to the machine or VM where AzCopy will be running. This copy of the certificate should be in .PFX or .PEM format, and must include the private key. The private key should be password-protected.

Type the following command, and then press the ENTER key.

```
export AZCOPY_AUTO_LOGIN_TYPE=SPN
export AZCOPY_SPA_CERT_PATH=<path-to-certificate-file>
export AZCOPY_SPA_CERT_PASSWORD=<certificate-password>
export AZCOPY_TENANT_ID=<tenant-id>
```

Replace the `<path-to-certificate-file>` placeholder with the relative or fully qualified path to the certificate file. AzCopy saves the path to this certificate but it doesn't save a copy of the certificate, so make sure to keep that

certificate in place. Replace the `<certificate-password>` placeholder with the password of the certificate. Replace the `<tenant-id>` placeholder with the tenant ID of the organization to which the storage account belongs. To find the tenant ID, select **Azure Active Directory > Properties > Directory ID** in the Azure portal.

**NOTE**

Consider using a prompt to collect the password from the user. That way, your password won't appear in your command history.

Then, run any azcopy command (For example: `azcopy list https://contoso.blob.core.windows.net`).

## Next steps

- For more information about AzCopy, [Get started with AzCopy](#)
- If you have questions, issues, or general feedback, submit them [on GitHub page](#).

# Optimize the performance of AzCopy with Azure Storage

8/22/2022 • 6 minutes to read • [Edit Online](#)

AzCopy is a command-line utility that you can use to copy blobs or files to or from a storage account. This article helps you to optimize performance.

## NOTE

If you're looking for content to help you get started with AzCopy, see [Get started with AzCopy](#)

You can benchmark performance, and then use commands and environment variables to find an optimal tradeoff between performance and resource consumption.

## Run benchmark tests

You can run a performance benchmark test on specific blob containers or file shares to view general performance statistics and to identify performance bottlenecks. You can run the test by uploading or downloading generated test data.

Use the following command to run a performance benchmark test.

### Syntax

```
azcopy benchmark 'https://<storage-account-name>.blob.core.windows.net/<container-name>'
```

### Example

```
azcopy benchmark 'https://mystorageaccount.blob.core.windows.net/mycontainer/myBlobDirectory?sv=2018-03-28&ss=bjqt&srs=sco&sp=rjk1hjup&se=2019-05-10T04:37:48Z&st=2019-05-09T20:37:48Z&spr=https&sig=%2FSOVEFfsKDqRry4bk3qz1vAQFwY5DDzp2%2B%2F3Eykf%2FJLs%3D'
```

## TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

This command runs a performance benchmark by uploading test data to a specified destination. The test data is generated in memory, uploaded to the destination, then deleted from the destination after the test is complete. You can specify how many files to generate and what size you'd like them to be by using optional command parameters.

If you prefer to run this test by downloading data, set the `mode` parameter to `download`. For detailed reference docs, see [azcopy benchmark](#).

## Optimize for large numbers of small files

Throughput can decrease when transferring small files, especially when transferring large numbers of them. To maximize performance, reduce the size of each job. For download and upload operations, increase concurrency,

decrease log activity, and turn off features that incur high performance costs.

#### Reduce the size of each job

To achieve optimal performance, ensure that each job transfers fewer than 10 million files. Jobs that transfer more than 50 million files can perform poorly because the AzCopy job tracking mechanism incurs a significant amount of overhead. To reduce overhead, consider dividing large jobs into smaller ones.

One way to reduce the size of a job is to limit the number of files affected by a job. You can use command parameters to do that. For example, a job can copy only a subset of directories by using the `include path` parameter as part of the [azcopy copy](#) command.

Use the `include-pattern` parameter to copy files that have a specific extension (for example: `*.pdf`). In a separate job, use the `exclude-pattern` parameter to copy all files that don't have `*.pdf` extension. See [Upload specific files](#) and [Download specific blobs](#) for examples.

After you've decided how to divide large jobs into smaller ones, consider running jobs on more than one Virtual Machine (VM).

#### Increase concurrency

If you're uploading or downloading files, use the `AZCOPY_CONCURRENCY_VALUE` environment variable to increase the number of concurrent requests that can occur on your machine. Set this variable as high as possible without compromising the performance of your machine. To learn more about this variable, see the [Increase the number of concurrent requests](#) section of this article.

If you're copying blobs between storage accounts, consider setting the value of the `AZCOPY_CONCURRENCY_VALUE` environment variable to a value greater than `1000`. You can set this variable high because AzCopy uses server-to-server APIs, so data is copied directly between storage servers and does not use your machine's processing power.

#### Decrease the number of logs generated

You can improve performance by reducing the number of log entries that AzCopy creates as it completes an operation. By default, AzCopy logs all activity related to an operation. To achieve optimal performance, consider setting the `log-level` parameter of your copy, sync, or remove command to `ERROR`. That way, AzCopy logs only errors. By default, the value log level is set to `INFO`.

#### Turn off length checking

If you're uploading or downloading files, consider setting the `--check-length` of your copy and sync commands to `false`. This prevents AzCopy from verifying the length of a file after a transfer. By default, AzCopy checks the length to ensure that source and destination files match after a transfer completes. AzCopy performs this check after each file transfer. This check can degrade performance when jobs transfer large numbers of small files.

#### Turn on concurrent local scanning (Linux)

File scans on some Linux systems don't execute fast enough to saturate all of the parallel network connections. In these cases, you can set the `AZCOPY_CONCURRENT_SCAN` to a higher number.

## Increase the number of concurrent requests

You can increase throughput by setting the `AZCOPY_CONCURRENCY_VALUE` environment variable. This variable specifies the number of concurrent requests that can occur.

If your computer has fewer than 5 CPUs, then the value of this variable is set to `32`. Otherwise, the default value is equal to 16 multiplied by the number of CPUs. The maximum default value of this variable is `300`, but you can manually set this value higher or lower.

OPERATING SYSTEM	COMMAND
Windows	<code>set AZCOPY_CONCURRENCY_VALUE=&lt;value&gt;</code>
Linux	<code>export AZCOPY_CONCURRENCY_VALUE=&lt;value&gt;</code>
macOS	<code>export AZCOPY_CONCURRENCY_VALUE=&lt;value&gt;</code>

Use the `azcopy env` to check the current value of this variable. If the value is blank, then you can read which value is being used by looking at the beginning of any AzCopy log file. The selected value, and the reason it was selected, are reported there.

Before you set this variable, we recommend that you run a benchmark test. The benchmark test process will report the recommended concurrency value. Alternatively, if your network conditions and payloads vary, set this variable to the word `AUTO` instead of to a particular number. That will cause AzCopy to always run the same automatic tuning process that it uses in benchmark tests.

## Limit the throughput data rate

You can use the `cap-mbps` flag in your commands to place a ceiling on the throughput data rate. For example, the following command resumes a job and caps throughput to `10` megabits (Mb) per second.

```
azcopy jobs resume <job-id> --cap-mbps 10
```

## Optimize memory use

Set the `AZCOPY_BUFFER_GB` environment variable to specify the maximum amount of your system memory you want AzCopy to use for buffering when downloading and uploading files. Express this value in gigabytes (GB).

OPERATING SYSTEM	COMMAND
Windows	<code>set AZCOPY_BUFFER_GB=&lt;value&gt;</code>
Linux	<code>export AZCOPY_BUFFER_GB=&lt;value&gt;</code>
macOS	<code>export AZCOPY_BUFFER_GB=&lt;value&gt;</code>

### NOTE

Job tracking always incurs additional overhead in memory usage. The amount varies based on the number of transfers in a job. Buffers are the largest component of memory usage. You can help control overhead by using `AZCOPY_BUFFER_GB` to approximately meet your requirements, but there is no flag available to strictly cap the overall memory usage.

## Optimize file synchronization

The `sync` command identifies all files at the destination, and then compares file names and last modified timestamps before the starting the sync operation. If you have a large number of files, then you can improve performance by eliminating this up-front processing.

To accomplish this, use the `azcopy copy` command instead, and set the `--overwrite` flag to `ifSourceNewer`. AzCopy will compare files as they are copied without performing any up-front scans and comparisons. This

provides a performance edge in cases where there are a large number of files to compare.

The [azcopy copy](#) command doesn't delete files from the destination, so if you want to delete files at the destination when they no longer exist at the source, then use the [azcopy sync](#) command with the

`--delete-destination` flag set to a value of `true` or `prompt`.

## See also

- [Get started with AzCopy](#)

# Find errors and resume jobs by using log and plan files in AzCopy

8/22/2022 • 3 minutes to read • [Edit Online](#)

AzCopy is a command-line utility that you can use to copy blobs or files to or from a storage account. This article helps you use logs to diagnose errors, and then use plan files to resume jobs. This article also shows how to configure log and plan files by changing their verbosity level, and the default location where they are stored.

## NOTE

If you're looking for content to help you get started with AzCopy, see [Get started with AzCopy](#). This article applies to AzCopy V10 as is this is the currently supported version of AzCopy. If you need to use a previous version of AzCopy, see [Use the previous version of AzCopy](#).

## Log and plan files

AzCopy creates *log* and *plan* files for every job. You can use these logs to investigate and troubleshoot any potential problems.

The logs will contain the status of failure (`UPLOADFAILED`, `COPYFAILED`, and `DOWNLOADFAILED`), the full path, and the reason of the failure.

By default, the log and plan files are located in the `%USERPROFILE%\azcopy` directory on Windows or `$HOME$\azcopy` directory on Mac and Linux, but you can change that location.

The relevant error isn't necessarily the first error that appears in the file. For errors such as network errors, timeouts and Server Busy errors, AzCopy will retry up to 20 times and usually the retry process succeeds. The first error that you see might be something harmless that was successfully retried. So instead of looking at the first error in the file, look for the errors that are near `UPLOADFAILED`, `COPYFAILED`, or `DOWNLOADFAILED`.

## IMPORTANT

When submitting a request to Microsoft Support (or troubleshooting the issue involving any third party), share the redacted version of the command you want to execute. This ensures the SAS isn't accidentally shared with anybody. You can find the redacted version at the start of the log file.

## Review the logs for errors

The following command will get all errors with `UPLOADFAILED` status from the `04dc9ca9-158f-7945-5933-564021086c79` log:

### Windows (PowerShell)

```
Select-String UPLOADFAILED .\04dc9ca9-158f-7945-5933-564021086c79.log
```

### Linux

```
grep UPLOADFAILED .\04dc9ca9-158f-7945-5933-564021086c79.log
```

## View and resume jobs

Each transfer operation will create an AzCopy job. Use the following command to view the history of jobs:

```
azcopy jobs list
```

To view the job statistics, use the following command:

```
azcopy jobs show <job-id>
```

To filter the transfers by status, use the following command:

```
azcopy jobs show <job-id> --with-status=Failed
```

**TIP**

The value of the `--with-status` flag is case-sensitive.

Use the following command to resume a failed/canceled job. This command uses its identifier along with the SAS token as it isn't persistent for security reasons:

```
azcopy jobs resume <job-id> --source-sas=<sas-token> --destination-sas=<sas-token>
```

**TIP**

Enclose path arguments such as the SAS token with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

When you resume a job, AzCopy looks at the job plan file. The plan file lists all the files that were identified for processing when the job was first created. When you resume a job, AzCopy will attempt to transfer all of the files that are listed in the plan file which weren't already transferred.

## Change the location of plan files

Use any of these commands.

OPERATING SYSTEM	COMMAND
Windows	PowerShell: <code>\$env:AZCOPY_JOB_PLAN_LOCATION=&lt;value&gt;</code> In a command prompt use: <code>set AZCOPY_JOB_PLAN_LOCATION=&lt;value&gt;</code>
Linux	<code>export AZCOPY_JOB_PLAN_LOCATION=&lt;value&gt;</code>
macOS	<code>export AZCOPY_JOB_PLAN_LOCATION=&lt;value&gt;</code>

Use the `azcopy env` to check the current value of this variable. If the value is blank, then plan files are written to the default location.

## Change the location of log files

Use any of these commands.

OPERATING SYSTEM	COMMAND
Windows	PowerShell: <code>\$env:AZCOPY_LOG_LOCATION=&lt;value&gt;</code> In a command prompt use: <code>set AZCOPY_LOG_LOCATION=&lt;value&gt;</code>
Linux	<code>export AZCOPY_LOG_LOCATION=&lt;value&gt;</code>
macOS	<code>export AZCOPY_LOG_LOCATION=&lt;value&gt;</code>

Use the `azcopy env` to check the current value of this variable. If the value is blank, then logs are written to the default location.

## Change the default log level

By default, AzCopy log level is set to `INFO`. If you would like to reduce the log verbosity to save disk space, overwrite this setting by using the `--log-level` option.

Available log levels are: `DEBUG`, `INFO`, `WARNING`, `ERROR`, and `NONE`.

## Remove plan and log files

If you want to remove all plan and log files from your local machine to save disk space, use the `azcopy jobs clean` command.

To remove the plan and log files associated with only one job, use `azcopy jobs rm <job-id>`. Replace the `<job-id>` placeholder in this example with the job ID of the job.

## See also

- [Get started with AzCopy](#)

# Upload files to Azure Blob storage by using AzCopy

8/22/2022 • 6 minutes to read • [Edit Online](#)

You can upload files and directories to Blob storage by using the AzCopy v10 command-line utility.

To see examples for other types of tasks such as downloading blobs, synchronizing with Blob storage, or copying blobs between accounts, see the links presented in the [Next Steps](#) section of this article.

## Get started

See the [Get started with AzCopy](#) article to download AzCopy and learn about the ways that you can provide authorization credentials to the storage service.

### NOTE

The examples in this article assume that you've provided authorization credentials by using Azure Active Directory (Azure AD).

If you'd rather use a SAS token to authorize access to blob data, then you can append that token to the resource URL in each AzCopy command. For example:

```
'https://<storage-account-name>.blob.core.windows.net/<container-name><SAS-token>'.
```

## Create a container

You can use the [azcopy make](#) command to create a container.

### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

### Syntax

```
azcopy make 'https://<storage-account-name>.<blob or dfs>.core.windows.net/<container-name>'
```

### Example

```
azcopy make 'https://mystorageaccount.blob.core.windows.net/mycontainer'
```

### Example (hierarchical namespace)

```
azcopy make 'https://mystorageaccount.dfs.core.windows.net/mycontainer'
```

For detailed reference docs, see [azcopy make](#).

## Upload a file

Upload a file by using the [azcopy copy](#) command.

**TIP**

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

## Syntax

```
azcopy copy '<local-file-path>' 'https://<storage-account-name>.<blob or dfs>.core.windows.net/<container-name>/<blob-name>'
```

## Example

```
azcopy copy 'C:\myDirectory\myTextFile.txt'
'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile.txt'
```

## Example (hierarchical namespace)

```
azcopy copy 'C:\myDirectory\myTextFile.txt'
'https://mystorageaccount.dfs.core.windows.net/mycontainer/myTextFile.txt'
```

You can also upload a file by using a wildcard symbol (\*) anywhere in the file path or file name. For example:

'C:\myDirectory\\*.txt' , or C:\my\*\\*.txt .

## Upload a directory

Upload a directory by using the [azcopy copy](#) command.

This example copies a directory (and all of the files in that directory) to a blob container. The result is a directory in the container by the same name.

**TIP**

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

## Syntax

```
azcopy copy '<local-directory-path>' 'https://<storage-account-name>.<blob or
dfs>.core.windows.net/<container-name>' --recursive
```

## Example

```
azcopy copy 'C:\myDirectory' 'https://mystorageaccount.blob.core.windows.net/mycontainer' --recursive
```

## Example (hierarchical namespace)

```
azcopy copy 'C:\myDirectory' 'https://mystorageaccount.dfs.core.windows.net/mycontainer' --recursive
```

To copy to a directory within the container, just specify the name of that directory in your command string.

## Example

```
azcopy copy 'C:\myDirectory' 'https://mystorageaccount.blob.core.windows.net/mycontainer/myBlobDirectory' --recursive
```

### Example (hierarchical namespace)

```
azcopy copy 'C:\myDirectory' 'https://mystorageaccount.dfs.core.windows.net/mycontainer/myBlobDirectory' --recursive
```

If you specify the name of a directory that doesn't exist in the container, AzCopy creates a new directory by that name.

## Upload directory contents

Upload the contents of a directory by using the [azcopy copy](#) command. Use the wildcard symbol (\*) to upload the contents without copying the containing directory itself.

#### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

### Syntax

```
azcopy copy '<local-directory-path>*' 'https://<storage-account-name>.<blob or
dfs>.core.windows.net/<container-name>/<directory-path>'
```

### Example

```
azcopy copy 'C:\myDirectory*' 'https://mystorageaccount.blob.core.windows.net/mycontainer/myBlobDirectory'
```

### Example (hierarchical namespace)

```
azcopy copy 'C:\myDirectory*' 'https://mystorageaccount.dfs.core.windows.net/mycontainer/myBlobDirectory'
```

Append the `--recursive` flag to upload files in all subdirectories.

## Upload specific files

You can upload specific files by using complete file names, partial names with wildcard characters (\*), or by using dates and times.

#### TIP

These examples enclose path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

### Specify multiple complete file names

Use the [azcopy copy](#) command with the `--include-path` option. Separate individual file names by using a semicolon ( ; ).

## Syntax

```
azcopy copy '<local-directory-path>' 'https://<storage-account-name>.<blob or
dfs>.core.windows.net/<container-name>' --include-path <semicolon-separated-file-list>
```

## Example

```
azcopy copy 'C:\myDirectory' 'https://mystorageaccount.blob.core.windows.net/mycontainer' --include-path
'photos;documents\myfile.txt' --recursive'
```

## Example (hierarchical namespace)

```
azcopy copy 'C:\myDirectory' 'https://mystorageaccount.dfs.core.windows.net/mycontainer' --include-path
'photos;documents\myfile.txt' --recursive'
```

In this example, AzCopy transfers the `C:\myDirectory\photos` directory and the `C:\myDirectory\documents\myfile.txt` file. Include the `--recursive` option to transfer all files in the `C:\myDirectory\photos` directory.

You can also exclude files by using the `--exclude-path` option. To learn more, see [azcopy copy](#) reference docs.

## Use wildcard characters

Use the [azcopy copy](#) command with the `--include-pattern` option. Specify partial names that include the wildcard characters. Separate names by using a semicolon (`;`).

## Syntax

```
azcopy copy '<local-directory-path>' 'https://<storage-account-name>.<blob or
dfs>.core.windows.net/<container-name>' --include-pattern <:semicolon-separated-file-list-with-wildcard-
characters>
```

## Example

```
azcopy copy 'C:\myDirectory' 'https://mystorageaccount.blob.core.windows.net/mycontainer' --include-pattern
'myfile*.txt;*.pdf*'
```

## Example (hierarchical namespace)

```
azcopy copy 'C:\myDirectory' 'https://mystorageaccount.dfs.core.windows.net/mycontainer' --include-pattern
'myfile*.txt;*.pdf*'
```

You can also exclude files by using the `--exclude-pattern` option. To learn more, see [azcopy copy](#) reference docs.

The `--include-pattern` and `--exclude-pattern` options apply only to filenames and not to the path. If you want to copy all of the text files that exist in a directory tree, use the `-recursive` option to get the entire directory tree, and then use the `--include-pattern` and specify `*.txt` to get all of the text files.

## Upload files that were modified before or after a date and time

Use the [azcopy copy](#) command with the `--include-before` or `--include-after` option. Specify a date and time in ISO-8601 format (For example: `2020-08-19T15:04:00Z` ).

The following examples upload files that were modified on or after the specified date.

## Syntax

```
azcopy copy '<local-directory-path>*' 'https://<storage-account-name>.<blob or
dfs>.core.windows.net/<container-or-directory-name>' --include-after <Date-Time-in-ISO-8601-format>
```

## Example

```
azcopy copy 'C:\myDirectory*' 'https://mystorageaccount.blob.core.windows.net/mycontainer/FileDirectory' --include-after '2020-08-19T15:04:00Z'
```

## Example (hierarchical namespace)

```
azcopy copy 'C:\myDirectory*' 'https://mystorageaccount.dfs.core.windows.net/mycontainer/FileDirectory' --include-after '2020-08-19T15:04:00Z'
```

For detailed reference, see the [azcopy copy](#) reference docs.

## Upload with index tags

You can upload a file and add [blob index tags\(preview\)](#) to the target blob.

If you're using Azure AD authorization, your security principal must be assigned the [Storage Blob Data Owner](#) role or it must be given permission to the

`Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write` [Azure resource provider operation](#) via a custom Azure role. If you're using a Shared Access Signature (SAS) token, that token must provide access to the blob's tags via the `t` SAS permission.

To add tags, use the `--blob-tags` option along with a URL encoded key-value pair. For example, to add the key `my tag` and a value `my tag value`, you would add `--blob-tags='my%20tag=my%20tag%20value'` to the destination parameter.

Separate multiple index tags by using an ampersand (`&`). For example, if you want to add a key `my second tag` and a value `my second tag value`, the complete option string would be

```
--blob-tags='my%20tag=my%20tag%20value&my%20second%20tag=my%20second%20tag%20value'.
```

The following examples show how to use the `--blob-tags` option.

### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

## Upload a file

```
azcopy copy 'C:\myDirectory\myTextFile.txt'
'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile.txt' --blob-
tags='my%20tag=my%20tag%20value&my%20second%20tag=my%20second%20tag%20value'
```

## Upload a directory

```
azcopy copy 'C:\myDirectory' 'https://mystorageaccount.blob.core.windows.net/mycontainer' --recursive --
blob-tags='my%20tag=my%20tag%20value&my%20second%20tag=my%20second%20tag%20value'
```

## Upload directory contents

```
azcopy copy 'C:\myDirectory*' 'https://mystorageaccount.blob.core.windows.net/mycontainer/myBlobDirectory'
--blob-tags='my%20tag=my%20tag%20value&my%20second%20tag=my%20second%20tag%20value'
```

#### NOTE

If you specify a directory for the source, all the blobs that are copied to the destination will have the same tags that you specify in the command.

## Upload with optional flags

You can tweak your upload operation by using optional flags. Here's a few examples.

SCENARIO	FLAG
Upload files as Append Blobs or Page Blobs.	--blob-type=[BlockBlob PageBlob AppendBlob]
Upload to a specific access tier (such as the archive tier).	--block-blob-tier=[None Hot Cool Archive]

For a complete list, see [options](#).

## Next steps

Find more examples in these articles:

- [Examples: Download](#)
- [Examples: Copy between accounts](#)
- [Examples: Synchronize](#)
- [Examples: Amazon S3 buckets](#)
- [Examples: Google Cloud Storage](#)
- [Examples: Azure Files](#)
- [Tutorial: Migrate on-premises data to cloud storage by using AzCopy](#)

See these articles to configure settings, optimize performance, and troubleshoot issues:

- [AzCopy configuration settings](#)
- [Optimize the performance of AzCopy](#)
- [Find errors and resume jobs by using log and plan files in AzCopy](#)
- [Troubleshoot problems with AzCopy v10](#)

# Download blobs from Azure Blob Storage by using AzCopy

8/22/2022 • 5 minutes to read • [Edit Online](#)

You can download blobs and directories from Blob storage by using the AzCopy v10 command-line utility.

To see examples for other types of tasks such as uploading files, synchronizing with Blob storage, or copying blobs between accounts, see the links presented in the [Next Steps](#) section of this article.

## Get started

See the [Get started with AzCopy](#) article to download AzCopy and learn about the ways that you can provide authorization credentials to the storage service.

### NOTE

The examples in this article assume that you've provided authorization credentials by using Azure Active Directory (Azure AD).

If you'd rather use a SAS token to authorize access to blob data, then you can append that token to the resource URL in each AzCopy command. For example:

```
'https://<storage-account-name>.blob.core.windows.net/<container-name><SAS-token>'.
```

## Download a blob

Download a blob by using the [azcopy copy](#) command.

### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

### Syntax

```
azcopy copy 'https://<storage-account-name>.<blob or dfs>.core.windows.net/<container-name>/<blob-path>' '<local-file-path>'
```

### Example

```
azcopy copy 'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile.txt' 'C:\myDirectory\myTextFile.txt'
```

### Example (hierarchical namespace)

```
azcopy copy 'https://mystorageaccount.dfs.core.windows.net/mycontainer/myTextFile.txt' 'C:\myDirectory\myTextFile.txt'
```

#### NOTE

If the `Content-md5` property value of a blob contains a hash, AzCopy calculates an MD5 hash for downloaded data and verifies that the MD5 hash stored in the blob's `Content-md5` property matches the calculated hash. If these values don't match, the download fails unless you override this behavior by appending `--check-md5=NoCheck` or `--check-md5=LogOnly` to the copy command.

## Download a directory

Download a directory by using the [azcopy copy](#) command.

#### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

### Syntax

```
azcopy copy 'https://<storage-account-name>.<blob or dfs>.core.windows.net/<container-name>/<directory-path>' '<local-directory-path>' --recursive
```

### Example

```
azcopy copy 'https://mystorageaccount.blob.core.windows.net/mycontainer/myBlobDirectory' 'C:\myDirectory' --recursive
```

### Example (hierarchical namespace)

```
azcopy copy 'https://mystorageaccount.dfs.core.windows.net/mycontainer/myBlobDirectory' 'C:\myDirectory' --recursive
```

This example results in a directory named `C:\myDirectory\myBlobDirectory` that contains all of the downloaded blobs.

## Download directory contents

You can download the contents of a directory without copying the containing directory itself by using the wildcard symbol (\*).

#### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

#### NOTE

Currently, this scenario is supported only for accounts that don't have a hierarchical namespace.

### Syntax

```
azcopy copy 'https://<storage-account-name>.blob.core.windows.net/<container-name>/*' '<local-directory-path>'
```

## Example

```
azcopy copy 'https://mystorageaccount.blob.core.windows.net/mycontainer/myBlobDirectory/*' 'C:\myDirectory'
```

Append the `--recursive` flag to download files in all subdirectories.

## Download specific blobs

You can download specific blobs by using complete file names, partial names with wildcard characters (\*), or by using dates and times.

### TIP

These examples enclose path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

### Specify multiple complete blob names

Use the `azcopy copy` command with the `--include-path` option. Separate individual blob names by using a semicolon ( ; ).

## Syntax

```
azcopy copy 'https://<storage-account-name>.<blob or dfs>.core.windows.net/<container-or-directory-name>' '<local-directory-path>' --include-path <semicolon-separated-file-list>
```

## Example

```
azcopy copy 'https://mystorageaccount.blob.core.windows.net/mycontainer/FileDirectory' 'C:\myDirectory' --include-path 'photos;documents\myFile.txt' --recursive
```

### Example (hierarchical namespace)

```
azcopy copy 'https://mystorageaccount.dfs.core.windows.net/mycontainer/FileDirectory' 'C:\myDirectory' --include-path 'photos;documents\myFile.txt' --recursive
```

In this example, AzCopy transfers the

`https://mystorageaccount.blob.core.windows.net/mycontainer/FileDirectory/photos` directory and the  
`https://mystorageaccount.blob.core.windows.net/mycontainer/FileDirectory/documents/myFile.txt` file. Include the  
`--recursive` option to transfer all blobs in the  
`https://mystorageaccount.blob.core.windows.net/mycontainer/FileDirectory/photos` directory.

You can also exclude blobs by using the `--exclude-path` option. To learn more, see [azcopy copy](#) reference docs.

### Use wildcard characters

Use the `azcopy copy` command with the `--include-pattern` option. Specify partial names that include the wildcard characters. Separate names by using a semicolon ( ; ).

## Syntax

```
azcopy copy 'https://<storage-account-name>.<blob or dfs>.core.windows.net/<container-or-directory-name>' '<local-directory-path>' --include-pattern <semicolon-separated-file-list-with-wildcard-characters>
```

## Example

```
azcopy copy 'https://mystorageaccount.blob.core.windows.net/mycontainer/FileDirectory' 'C:\myDirectory' --include-pattern 'myFile*.txt;*.pdf'
```

## Example (hierarchical namespace)

```
azcopy copy 'https://mystorageaccount.dfs.core.windows.net/mycontainer/FileDirectory' 'C:\myDirectory' --include-pattern 'myFile*.txt;*.pdf'
```

You can also exclude blobs by using the `--exclude-pattern` option. To learn more, see [azcopy copy](#) reference docs.

The `--include-pattern` and `--exclude-pattern` options apply only to blob names and not to the path. If you want to copy all of the text files (blobs) that exist in a directory tree, use the `-recursive` option to get the entire directory tree, and then use the `-include-pattern` and specify `*.txt` to get all of the text files.

### Download blobs that were modified before or after a date and time

Use the [azcopy copy](#) command with the `--include-before` or `--include-after` option. Specify a date and time in ISO-8601 format (For example: `2020-08-19T15:04:00Z`).

The following examples download files that were modified on or after the specified date.

## Syntax

```
azcopy copy 'https://<storage-account-name>.<blob or dfs>.core.windows.net/<container-or-directory-name>/*' '<local-directory-path>' --include-after <Date-Time-in-ISO-8601-format>
```

## Example

```
azcopy copy 'https://mystorageaccount.blob.core.windows.net/mycontainer/FileDirectory/*' 'C:\myDirectory' --include-after '2020-08-19T15:04:00Z'
```

## Example (hierarchical namespace)

```
azcopy copy 'https://mystorageaccount.dfs.core.windows.net/mycontainer/FileDirectory/*' 'C:\myDirectory' --include-after '2020-08-19T15:04:00Z'
```

For detailed reference, see the [azcopy copy](#) reference docs.

### Download previous versions of a blob

If you've enabled [blob versioning](#), you can download one or more previous versions of a blob.

First, create a text file that contains a list of [version IDs](#). Each version ID must appear on a separate line. For example:

```
2020-08-17T05:50:34.2199403Z
2020-08-17T05:50:34.5041365Z
2020-08-17T05:50:36.7607103Z
```

Then, use the [azcopy copy](#) command with the `--list-of-versions` option. Specify the location of the text file that contains the list of versions (For example: `D:\\list-of-versions.txt`).

### Download a blob snapshot

You can download a [blob snapshot](#) by referencing the [DateTime](#) value of a blob snapshot.

## Syntax

```
azcopy copy 'https://<storage-account-name>.<blob or dfs>.core.windows.net/<container-name>/<blob-path>?sharesnapshot=<DateTime-of-snapshot>' '<local-file-path>'
```

## Example

```
azcopy copy 'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile.txt?sharesnapshot=2020-09-23T08:21:07.000000Z' 'C:\myDirectory\myTextFile.txt'
```

## Example (hierarchical namespace)

```
azcopy copy 'https://mystorageaccount.dfs.core.windows.net/mycontainer/myTextFile.txt?sharesnapshot=2020-09-23T08:21:07.000000Z' 'C:\myDirectory\myTextFile.txt'
```

### NOTE

If you are using a SAS token to authorize access to blob data, then append snapshot **DateTime** after the SAS token. For example:

```
'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile.txt?sv=2018-03-28&ss=bjqs&srs=sco&sp=rjk1hjup&se=2019-05-10T04:37:48Z&st=2019-05-09T20:37:48Z&spr=https&sig=%2F5OVEFFsKDqRry4bk3qz1vAQFWY5DDzp2%2B%2F3Eykf%2FJLs%3D&sharesnapshot=2020-09-23T08:21:07.000000Z'
```

## Download with optional flags

You can tweak your download operation by using optional flags. Here's a few examples.

SCENARIO	FLAG
Automatically decompress files.	--decompress
Specify how detailed you want your copy-related log entries to be.	--log-level=[WARNING ERROR INFO NONE]
Specify if and how to overwrite the conflicting files and blobs at the destination.	--overwrite=[true false ifSourceNewer prompt]

For a complete list, see [options](#).

## Next steps

Find more examples in these articles:

- [Examples: Upload](#)
- [Examples: Copy between account](#)
- [Examples: Synchronize](#)
- [Examples: Amazon S3 buckets](#)
- [Examples: Google Cloud Storage](#)
- [Examples: Azure Files](#)
- [Tutorial: Migrate on-premises data to cloud storage by using AzCopy](#)

See these articles to configure settings, optimize performance, and troubleshoot issues:

- [AzCopy configuration settings](#)
- [Optimize the performance of AzCopy](#)

- Find errors and resume jobs by using log and plan files in AzCopy
- Troubleshoot problems with AzCopy v10

# Copy blobs between Azure storage accounts by using AzCopy

8/22/2022 • 5 minutes to read • [Edit Online](#)

You can copy blobs, directories, and containers between storage accounts by using the AzCopy v10 command-line utility.

To see examples for other types of tasks such as uploading files, downloading blobs, and synchronizing with Blob storage, see the links presented in the [Next Steps](#) section of this article.

AzCopy uses [server-to-server APIs](#), so data is copied directly between storage servers. These copy operations don't use the network bandwidth of your computer.

## Get started

See the [Get started with AzCopy](#) article to download AzCopy and learn about the ways that you can provide authorization credentials to the storage service.

### NOTE

The examples in this article assume that you've provided authorization credentials by using Azure Active Directory (Azure AD) and that your Azure AD identity has the proper role assignments for both source and destination accounts.

Alternatively, you can append a SAS token to either the source or destination URL in each AzCopy command. For example:

```
azcopy copy 'https://<source-storage-account-name>.blob.core.windows.net/<container-name>/<blob-path><SAS-token>' 'https://<destination-storage-account-name>.blob.core.windows.net/<container-name>/<blob-path><SAS-token>'
```

## Guidelines

Apply the following guidelines to your AzCopy commands.

- Your client must have network access to both the source and destination storage accounts. To learn how to configure the network settings for each storage account, see [Configure Azure Storage firewalls and virtual networks](#).
- If you copy to a premium block blob storage account, omit the access tier of a blob from the copy operation by setting the `s2s-preserve-access-tier` to `false` (For example: `--s2s-preserve-access-tier=false`). Premium block blob storage accounts don't support access tiers.
- If you copy to or from an account that has a hierarchical namespace, use `blob.core.windows.net` instead of `dfs.core.windows.net` in the URL syntax. [Multi-protocol access on Data Lake Storage](#) enables you to use `blob.core.windows.net`, and it is the only supported syntax for account to account copy scenarios.
- You can increase the throughput of copy operations by setting the value of the `AZCOPY_CONCURRENCY_VALUE` environment variable. To learn more, see [Increase Concurrency](#).
- If the source blobs have index tags, and you want to retain those tags, you'll have to reapply them to the destination blobs. For information about how to set index tags, see the [Copy blobs to another storage account with index tags](#) section of this article.

## Copy a blob

Copy a blob to another storage account by using the [azcopy copy](#) command.

### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

### Syntax

```
azcopy copy 'https://<source-storage-account-name>.blob.core.windows.net/<container-name>/<blob-path>'
'https://<destination-storage-account-name>.blob.core.windows.net/<container-name>/<blob-path>'
```

### Example

```
azcopy copy 'https://mysourceaccount.blob.core.windows.net/mycontainer/myTextFile.txt'
'https://mydestinationaccount.blob.core.windows.net/mycontainer/myTextFile.txt'
```

The copy operation is synchronous so when the command returns, that indicates that all files have been copied.

## Copy a directory

Copy a directory to another storage account by using the [azcopy copy](#) command.

### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

### Syntax

```
azcopy copy 'https://<source-storage-account-name>.blob.core.windows.net/<container-name>/<directory-path>'
'https://<destination-storage-account-name>.blob.core.windows.net/<container-name>' --recursive
```

### Example

```
azcopy copy 'https://mysourceaccount.blob.core.windows.net/mycontainer/myBlobDirectory'
'https://mydestinationaccount.blob.core.windows.net/mycontainer' --recursive
```

The copy operation is synchronous so when the command returns, that indicates that all files have been copied.

## Copy a container

Copy a container to another storage account by using the [azcopy copy](#) command.

### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

### Syntax

```
azcopy copy 'https://<source-storage-account-name>.blob.core.windows.net/<container-name>'
'https://<destination-storage-account-name>.blob.core.windows.net/<container-name>' --recursive
```

## Example

```
azcopy copy 'https://mysourceaccount.blob.core.windows.net/mycontainer'
'https://mydestinationaccount.blob.core.windows.net/mycontainer' --recursive
```

The copy operation is synchronous so when the command returns, that indicates that all files have been copied.

## Copy containers, directories, and blobs

Copy all containers, directories, and blobs to another storage account by using the [azcopy copy](#) command.

### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

## Syntax

```
azcopy copy 'https://<source-storage-account-name>.blob.core.windows.net/' 'https://<destination-storage-
account-name>.blob.core.windows.net/' --recursive
```

## Example

```
azcopy copy 'https://mysourceaccount.blob.core.windows.net/'
'https://mydestinationaccount.blob.core.windows.net' --recursive
```

The copy operation is synchronous so when the command returns, that indicates that all files have been copied.

## Copy blobs and add index tags

Copy blobs to another storage account and add [blob index tags\(preview\)](#) to the target blob.

If you're using Azure AD authorization, your security principal must be assigned the [Storage Blob Data Owner](#) role or it must be given permission to the

`Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write` [Azure resource provider operation](#) via a custom Azure role. If you're using a Shared Access Signature (SAS) token, that token must provide access to the blob's tags via the `t` SAS permission.

To add tags, use the `--blob-tags` option along with a URL encoded key-value pair.

For example, to add the key `my tag` and a value `my tag value`, you would add

```
--blob-tags='my%20tag=my%20tag%20value'
```

Separate multiple index tags by using an ampersand (&). For example, if you want to add a key `my second tag` and a value `my second tag value`, the complete option string would be

```
--blob-tags='my%20tag=my%20tag%20value&my%20second%20tag=my%20second%20tag%20value'.
```

The following examples show how to use the `--blob-tags` option.

**TIP**

These examples enclose path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

## Blob example

```
azcopy copy 'https://mysourceaccount.blob.core.windows.net/mycontainer/myTextFile.txt'
'https://mydestinationaccount.blob.core.windows.net/mycontainer/myTextFile.txt' --blob-
tags='my%20tag=my%20tag%20value&my%20second%20tag=my%20second%20tag%20value'
```

## Directory example

```
azcopy copy 'https://mysourceaccount.blob.core.windows.net/mycontainer/myBlobDirectory'
'https://mydestinationaccount.blob.core.windows.net/mycontainer' --recursive --blob-
tags='my%20tag=my%20tag%20value&my%20second%20tag=my%20second%20tag%20value'
```

## Container example

```
azcopy copy 'https://mysourceaccount.blob.core.windows.net/mycontainer'
'https://mydestinationaccount.blob.core.windows.net/mycontainer' --recursive --blob-
tags='my%20tag=my%20tag%20value&my%20second%20tag=my%20second%20tag%20value'
```

## Account example

```
azcopy copy 'https://mysourceaccount.blob.core.windows.net/'
'https://mydestinationaccount.blob.core.windows.net' --recursive --blob-
tags='my%20tag=my%20tag%20value&my%20second%20tag=my%20second%20tag%20value'
```

The copy operation is synchronous so when the command returns, that indicates that all files have been copied.

**NOTE**

If you specify a directory, container, or account for the source, all the blobs that are copied to the destination will have the same tags that you specify in the command.

## Copy with optional flags

You can tweak your copy operation by using optional flags. Here's a few examples.

SCENARIO	FLAG
Copy blobs as Block, Page, or Append Blobs.	--blob-type=[BlockBlob PageBlob AppendBlob]
Copy to a specific access tier (such as the archive tier).	--block-blob-tier=[None Hot Cool Archive]
Automatically decompress files.	--decompress=[gzip deflate]

For a complete list, see [options](#).

## Next steps

Find more examples in these articles:

- [Examples: Upload](#)
- [Examples: Download](#)
- [Examples: Synchronize](#)
- [Examples: Amazon S3 buckets](#)
- [Examples: Google Cloud Storage](#)
- [Examples: Azure Files](#)
- [Tutorial: Migrate on-premises data to cloud storage by using AzCopy](#)

See these articles to configure settings, optimize performance, and troubleshoot issues:

- [AzCopy configuration settings](#)
- [Optimize the performance of AzCopy](#)
- [Find errors and resume jobs by using log and plan files in AzCopy](#)
- [Troubleshoot problems with AzCopy v10](#)

# Synchronize with Azure Blob storage by using AzCopy

8/22/2022 • 5 minutes to read • [Edit Online](#)

You can synchronize local storage with Azure Blob storage by using the AzCopy v10 command-line utility.

You can synchronize the contents of a local file system with a blob container. You can also synchronize containers and virtual directories with one another. Synchronization is one way. In other words, you choose which of these two endpoints is the source and which one is the destination. Synchronization also uses server to server APIs. The examples presented in this section also work with accounts that have a hierarchical namespace.

## NOTE

The current release of AzCopy doesn't synchronize between other sources and destinations (For example: File storage or Amazon Web Services (AWS) S3 buckets).

To see examples for other types of tasks such as uploading files, downloading blobs, or copying blobs between accounts, see the links presented in the [Next Steps](#) section of this article.

## Get started

See the [Get started with AzCopy](#) article to download AzCopy and learn about the ways that you can provide authorization credentials to the storage service.

## NOTE

The examples in this article assume that you've provided authorization credentials by using Azure Active Directory (Azure AD).

If you'd rather use a SAS token to authorize access to blob data, then you can append that token to the resource URL in each AzCopy command. For example:

```
'https://<storage-account-name>.blob.core.windows.net/<container-name><SAS-token>' .
```

## Guidelines

- The `sync` command compares file names and last modified timestamps. Set the `--delete-destination` optional flag to a value of `true` or `prompt` to delete files in the destination directory if those files no longer exist in the source directory.
- If you set the `--delete-destination` flag to `true`, AzCopy deletes files without providing a prompt. If you want a prompt to appear before AzCopy deletes a file, set the `--delete-destination` flag to `prompt`.
- If you plan to set the `--delete-destination` flag to `prompt` or `false`, consider using the `copy` command instead of the `sync` command and set the `--overwrite` parameter to `ifSourceNewer`. The `copy` command consumes less memory and incurs less billing costs because a copy operation doesn't have to index the source or destination prior to moving files.
- To prevent accidental deletions, make sure to enable the `soft delete` feature before you use the `--delete-destination=prompt|true` flag.

- The machine on which you run the sync command should have an accurate system clock because the last modified times are critical in determining whether a file should be transferred. If your system has significant clock skew, avoid modifying files at the destination too close to the time that you plan to run a sync command.

## Update a container with changes to a local file system

In this case, the container is the destination, and the local file system is the source.

### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

### Syntax

```
azcopy sync '<local-directory-path>' 'https://<storage-account-name>.blob.core.windows.net/<container-name>' --recursive
```

### Example

```
azcopy sync 'C:\myDirectory' 'https://mystorageaccount.blob.core.windows.net/mycontainer' --recursive
```

## Update a local file system with changes to a container

In this case, the local file system is the destination, and the container is the source.

### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

### Syntax

```
azcopy sync 'https://<storage-account-name>.blob.core.windows.net/<container-name>' 'C:\myDirectory' --recursive
```

### Example

```
azcopy sync 'https://mystorageaccount.blob.core.windows.net/mycontainer' 'C:\myDirectory' --recursive
```

## Update a container with changes in another container

The first container that appears in this command is the source. The second one is the destination. Make sure to append a SAS token to each source URL.

If you provide authorization credentials by using Azure Active Directory (Azure AD), you can omit the SAS token only from the destination URL. Make sure that you've set up the proper roles in your destination account. See [Option 1: Use Azure Active Directory](#).

**TIP**

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

## Syntax

```
azcopy sync 'https://<source-storage-account-name>.blob.core.windows.net/<container-name>/<SAS-token>'
'https://<destination-storage-account-name>.blob.core.windows.net/<container-name>' --recursive
```

## Example

```
azcopy sync 'https://mysourceaccount.blob.core.windows.net/mycontainer?sv=2018-03-
28&ss=bfqt&srt=sco&sp=rwdlacup&se=2019-07-04T05:30:08Z&st=2019-07-
03T21:30:08Z&spr=https&sig=CAFhgnc9gdGktvB=ska7bAiqIdM845yiyFwdMH481QA8%3D'
'https://mydestinationaccount.blob.core.windows.net/mycontainer' --recursive
```

## Update a directory with changes to a directory in another container

The first directory that appears in this command is the source. The second one is the destination. Make sure to append a SAS token to each source URL.

If you provide authorization credentials by using Azure Active Directory (Azure AD), you can omit the SAS token only from the destination URL. Make sure that you've set up the proper roles in your destination account. See [Option 1: Use Azure Active Directory](#).

**TIP**

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

## Syntax

```
azcopy sync 'https://<source-storage-account-name>.blob.core.windows.net/<container-name>/<directory-
name>/<SAS-token>' 'https://<destination-storage-account-name>.blob.core.windows.net/<container-
name>/<directory-name>' --recursive
```

## Example

```
azcopy sync 'https://mysourceaccount.blob.core.windows.net/<container-name>/myDirectory?sv=2018-03-
28&ss=bfqt&srt=sco&sp=rwdlacup&se=2019-07-04T05:30:08Z&st=2019-07-
03T21:30:08Z&spr=https&sig=CAFhgnc9gdGktvB=ska7bAiqIdM845yiyFwdMH481QA8%3D'
'https://mydestinationaccount.blob.core.windows.net/mycontainer/myDirectory' --recursive
```

## Synchronize with optional flags

You can tweak your sync operation by using optional flags. Here's a few examples.

SCENARIO	FLAG
Specify how strictly MD5 hashes should be validated when downloading.	--check-md5= [NoCheck LogOnly FailIfDifferent FailIfDifferentOrMissing]

SCENARIO	FLAG
Exclude files based on a pattern.	--exclude-path
Specify how detailed you want your sync-related log entries to be.	--log-level=[WARNING ERROR INFO NONE]

For a complete list of flags, see [options](#).

#### NOTE

The `--recursive` flag is set to `true` by default. The `--exclude-pattern` and `--include-pattern` flags apply to only file names and not other parts of the file path.

## Next steps

Find more examples in these articles:

- [Examples: Upload](#)
- [Examples: Download](#)
- [Examples: Copy between accounts](#)
- [Examples: Amazon S3 buckets](#)
- [Examples: Google Cloud Storage](#)
- [Examples: Azure Files](#)
- [Tutorial: Migrate on-premises data to cloud storage by using AzCopy](#)

See these articles to configure settings, optimize performance, and troubleshoot issues:

- [AzCopy configuration settings](#)
- [Optimize the performance of AzCopy](#)
- [Find errors and resume jobs by using log and plan files in AzCopy](#)
- [Troubleshoot problems with AzCopy v10](#)

# Replace blob properties and metadata by using AzCopy v10 (preview)

8/22/2022 • 3 minutes to read • [Edit Online](#)

You can use AzCopy to change the [access tier](#) of one or more blobs and replace (*overwrite*) the metadata, and index tags of one or more blobs.

## IMPORTANT

This capability is currently in PREVIEW. See the [Supplemental Terms of Use for Microsoft Azure Previews](#) for legal terms that apply to Azure features that are in beta, preview, or otherwise not yet released into general availability.

## Get started

See the [Get started with AzCopy](#) article to download AzCopy and learn about the ways that you can provide authorization credentials to the storage service.

## NOTE

The examples in this article assume that you've provided authorization credentials by using Azure Active Directory (Azure AD).

If you'd rather use a SAS token to authorize access to blob data, then you can append that token to the resource URL in each AzCopy command. For example:

```
'https://<storage-account-name>.blob.core.windows.net/<container-name><SAS-token>' .
```

## Change the access tier

To change the access tier of a blob, use the [azcopy set-properties](#) command and set the `-block-blob-tier` parameter to the name of the access tier.

## TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

## Syntax

```
azcopy set-properties 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<blob-name>' --block-blob-tier=<access-tier>
```

## Example

```
azcopy set-properties 'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile.txt' --block-blob-tier=hot
```

To change the access tier for all blobs in a virtual directory, refer to the virtual directory name instead of the

blob name, and then append `--recursive=true` to the command.

## Example

```
azcopy set-properties 'https://mystorageaccount.blob.core.windows.net/mycontainer/myvirtualdirectory' --block-blob-tier=hot --recursive=true
```

To *rehydrate* a blob from the archive tier to an online tier, set the `--rehydrate-priority` to `standard` or `high`. By default, this parameter is set to `standard`. To learn more about the trade offs of each option, see [Rehydration priority](#).

## Example

```
azcopy set-properties 'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile.txt' --block-blob-tier=hot --rehydrate-priority=high
```

# Replace metadata

To replace the metadata of a blob, use the [azcopy set-properties](#) command and set the `--metadata` parameter to one or more key-value pairs.

### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

## Syntax

```
azcopy set-properties 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<blob-name>' --metadata=<key>=<value>;<key>=<value>
```

## Example

```
azcopy set-properties 'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile.txt' --metadata=mykey1=myvalue1;mykey2=myvalue2
```

To replace the metadata for all blobs in a virtual directory, refer to the virtual directory name instead of the blob name, and then append `--recursive=true` to the command.

## Example

```
azcopy set-properties 'https://mystorageaccount.blob.core.windows.net/mycontainer/myvirtualdirectory' --metadata=mykey1=myvalue1;mykey2=myvalue2 --recursive=true
```

To clear metadata, omit the tags and append `--metadata=clear` to the end of the command.

## Example

```
azcopy set-properties 'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile.txt' --metadata=clear
```

# Replace index tags

To replace the index tags of a blob, use the [azcopy set-properties](#) command and set the `--blob-tags` parameter to one or more key-value pairs. Setting blob index tags can be performed by the [Storage Blob Data Owner](#) and by anyone with a Shared Access Signature that has permission to access the blob's tags (the `t` SAS permission). In addition, RBAC users with the `Microsoft.Storage/storageAccounts/blobServices/containers/blobs/tags/write` permission can perform this operation.

#### TIP

This example encloses path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

## Syntax

```
azcopy set-properties 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<blob-name>' --blob-tags=<tag>=<value>;<tag>=<value>
```

## Example

```
azcopy set-properties 'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile.txt' --blob-tags=mytag1=mytag1value;mytag2=mytag2value
```

To replace the index tags for all blobs in a virtual directory, refer to the virtual directory name instead of the blob name, and then append `--recursive=true` to the command.

## Example

```
azcopy set-properties 'https://mystorageaccount.blob.core.windows.net/mycontainer/myvirtualdirectory' --blob-tags=mytag1=mytag1value;mytag2=mytag2value
```

## Next steps

Find more examples in these articles:

- [Examples: Upload](#)
- [Examples: Download](#)
- [Examples: Copy between accounts](#)
- [Examples: Synchronize](#)
- [Examples: Amazon S3 buckets](#)
- [Examples: Google Cloud Storage](#)
- [Examples: Azure Files](#)
- [Tutorial: Migrate on-premises data to cloud storage by using AzCopy](#)

See these articles to configure settings, optimize performance, and troubleshoot issues:

- [AzCopy configuration settings](#)
- [Optimize the performance of AzCopy](#)
- [Find errors and resume jobs by using log and plan files in AzCopy](#)
- [Troubleshoot problems with AzCopy v10](#)

# Copy data from Amazon S3 to Azure Storage by using AzCopy

8/22/2022 • 5 minutes to read • [Edit Online](#)

AzCopy is a command-line utility that you can use to copy blobs or files to or from a storage account. This article helps you copy objects, directories, and buckets from Amazon Web Services (AWS) S3 to Azure Blob Storage by using AzCopy.

## Choose how you'll provide authorization credentials

- To authorize with the Azure Storage, use Azure Active Directory (AD) or a Shared Access Signature (SAS) token.
- To authorize with AWS S3, use an AWS access key and a secret access key.

### Authorize with Azure Storage

See the [Get started with AzCopy](#) article to download AzCopy, and choose how you'll provide authorization credentials to the storage service.

#### NOTE

The examples in this article assume that you've authenticated your identity by using the `AzCopy login` command. AzCopy then uses your Azure AD account to authorize access to data in Blob storage.

If you'd rather use a SAS token to authorize access to blob data, then you can append that token to the resource URL in each AzCopy command.

For example: `https://mystorageaccount.blob.core.windows.net/mycontainer?<SAS-token>`.

### Authorize with AWS S3

Gather your AWS access key and secret access key, and then set these environment variables:

OPERATING SYSTEM	COMMAND
Windows	<code>set AWS_ACCESS_KEY_ID=&lt;access-key&gt;</code> <code>set AWS_SECRET_ACCESS_KEY=&lt;secret-access-key&gt;</code>
Linux	<code>export AWS_ACCESS_KEY_ID=&lt;access-key&gt;</code> <code>export AWS_SECRET_ACCESS_KEY=&lt;secret-access-key&gt;</code>
macOS	<code>export AWS_ACCESS_KEY_ID=&lt;access-key&gt;</code> <code>export AWS_SECRET_ACCESS_KEY=&lt;secret-access-key&gt;</code>

## Copy objects, directories, and buckets

AzCopy uses the [Put Block From URL](#) API, so data is copied directly between AWS S3 and storage servers. These copy operations don't use the network bandwidth of your computer.

## TIP

The examples in this section enclose path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

These examples also work with accounts that have a hierarchical namespace. [Multi-protocol access on Data Lake Storage](#) enables you to use the same URL syntax (`blob.core.windows.net`) on those accounts.

## Copy an object

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

### Syntax

```
azcopy copy 'https://s3.amazonaws.com/<bucket-name>/<object-name>' 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<blob-name>'
```

### Example

```
azcopy copy 'https://s3.amazonaws.com/mybucket/myobject'
'https://mystorageaccount.blob.core.windows.net/mycontainer/myblob'
```

## NOTE

Examples in this article use path-style URLs for AWS S3 buckets (For example:

`http://s3.amazonaws.com/<bucket-name>` ).

You can also use virtual hosted-style URLs as well (For example: `http://bucket.s3.amazonaws.com` ).

To learn more about virtual hosting of buckets, see [Virtual Hosting of Buckets](#).

## Copy a directory

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

### Syntax

```
azcopy copy 'https://s3.amazonaws.com/<bucket-name>/<directory-name>' 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<directory-name>' --recursive=true
```

### Example

```
azcopy copy 'https://s3.amazonaws.com/mybucket/mydirectory'
'https://mystorageaccount.blob.core.windows.net/mycontainer/mydirectory' --recursive=true
```

## NOTE

This example appends the `--recursive` flag to copy files in all sub-directories.

## Copy the contents of a directory

You can copy the contents of a directory without copying the containing directory itself by using the wildcard symbol (\*).

### Syntax

```
azcopy copy 'https://s3.amazonaws.com/<bucket-name>/<directory-name>/*' 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<directory-name>' --recursive=true
```

## Example

```
azcopy copy 'https://s3.amazonaws.com/mybucket/mydirectory/*'
'https://mystorageaccount.blob.core.windows.net/mycontainer/mydirectory' --recursive=true
```

### Copy a bucket

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

#### Syntax

```
azcopy copy 'https://s3.amazonaws.com/<bucket-name>' 'https://<storage-account-name>.blob.core.windows.net/<container-name>' --recursive=true
```

### Example

```
azcopy copy 'https://s3.amazonaws.com/mybucket' 'https://mystorageaccount.blob.core.windows.net/mycontainer'
--recursive=true
```

### Copy all buckets in all regions

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

#### Syntax

```
azcopy copy 'https://s3.amazonaws.com/' 'https://<storage-account-name>.blob.core.windows.net' --
recursive=true
```

### Example

```
azcopy copy 'https://s3.amazonaws.com' 'https://mystorageaccount.blob.core.windows.net' --recursive=true
```

### Copy all buckets in a specific S3 region

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

#### Syntax

```
azcopy copy 'https://s3-<region-name>.amazonaws.com/' 'https://<storage-account-name>.blob.core.windows.net'
--recursive=true
```

### Example

```
azcopy copy 'https://s3-rds.eu-north-1.amazonaws.com' 'https://mystorageaccount.blob.core.windows.net' --
recursive=true
```

## Handle differences in object naming rules

AWS S3 has a different set of naming conventions for bucket names as compared to Azure blob containers. You can read about them [here](#). If you choose to copy a group of buckets to an Azure storage account, the copy operation might fail because of naming differences.

AzCopy handles two of the most common issues that can arise; buckets that contain periods and buckets that contain consecutive hyphens. AWS S3 bucket names can contain periods and consecutive hyphens, but a container in Azure can't. AzCopy replaces periods with hyphens and consecutive hyphens with a number that represents the number of consecutive hyphens (For example: a bucket named `my----bucket` becomes `my-4-bucket`).

Also, as AzCopy copies over files, it checks for naming collisions and attempts to resolve them. For example, if

there are buckets with the name `bucket-name` and `bucket.name`, AzCopy resolves a bucket named `bucket.name` first to `bucket-name` and then to `bucket-name-2`.

## Handle differences in object metadata

AWS S3 and Azure allow different sets of characters in the names of object keys. You can read about the characters that AWS S3 uses [here](#). On the Azure side, blob object keys adhere to the naming rules for [C# identifiers](#).

As part of an AzCopy `copy` command, you can provide a value for optional the `s2s-handle-invalid-metadata` flag that specifies how you would like to handle files where the metadata of the file contains incompatible key names. The following table describes each flag value.

FLAG VALUE	DESCRIPTION
<code>ExcludeIfInvalid</code>	(Default option) The metadata isn't included in the transferred object. AzCopy logs a warning.
<code>FailIfInvalid</code>	Objects aren't copied. AzCopy logs an error and includes that error in the failed count that appears in the transfer summary.
<code>RenameIfInvalid</code>	AzCopy resolves the invalid metadata key, and copies the object to Azure using the resolved metadata key value pair. To learn exactly what steps AzCopy takes to rename object keys, see the <a href="#">How AzCopy renames object keys</a> section below. If AzCopy is unable to rename the key, then the object won't be copied.

### How AzCopy renames object keys

AzCopy performs these steps:

1. Replaces invalid characters with '\_'.  
2. Adds the string `rename_` to the beginning of a new valid key.

This key will be used to save the original metadata **value**.

3. Adds the string `rename_key_` to the beginning of a new valid key. This key will be used to save original metadata **invalid key**. You can use this key to try to recover the metadata in Azure side since metadata key is preserved as a value on the Blob storage service.

## Next steps

Find more examples in these articles:

- [Examples: Upload](#)
- [Examples: Download](#)
- [Examples: Copy between accounts](#)
- [Examples: Synchronize](#)
- [Examples: Google Cloud Storage](#)
- [Examples: Azure Files](#)
- [Tutorial: Migrate on-premises data to cloud storage by using AzCopy](#)

See these articles to configure settings, optimize performance, and troubleshoot issues:

- AzCopy configuration settings
- Optimize the performance of AzCopy
- Troubleshoot AzCopy V10 issues in Azure Storage by using log files

# Copy data from Google Cloud Storage to Azure Storage by using AzCopy

8/22/2022 • 5 minutes to read • [Edit Online](#)

AzCopy is a command-line utility that you can use to copy blobs or files to or from a storage account. This article helps you copy objects, directories, and buckets from Google Cloud Storage to Azure Blob Storage by using AzCopy.

## Choose how you'll provide authorization credentials

- To authorize with Azure Storage, use Azure Active Directory (AD) or a Shared Access Signature (SAS) token.
- To authorize with Google Cloud Storage, use a service account key.

### Authorize with Azure Storage

See the [Get started with AzCopy](#) article to download AzCopy and learn about the ways that you can provide authorization credentials to the storage service.

#### NOTE

The examples in this article assume that you've provided authorization credentials by using Azure Active Directory (Azure AD).

If you'd rather use a SAS token to authorize access to blob data, then you can append that token to the resource URL in each AzCopy command. For example:

```
'https://<storage-account-name>.blob.core.windows.net/<container-name><SAS-token>' .
```

### Authorize with Google Cloud Storage

To authorize with Google Cloud Storage, you'll use a service account key. For information about how to create a service account key, see [Creating and managing service account keys](#).

After you've obtained a service key, set the `GOOGLE_APPLICATION_CREDENTIALS` environment variable to absolute path to the service account key file:

OPERATING SYSTEM	COMMAND
Windows	<pre>set GOOGLE_APPLICATION_CREDENTIALS=&lt;path-to-service-account-key&gt;</pre>
Linux	<pre>export GOOGLE_APPLICATION_CREDENTIALS=&lt;path-to-service-account-key&gt;</pre>
macOS	<pre>export GOOGLE_APPLICATION_CREDENTIALS=&lt;path-to-service-account-key&gt;</pre>

## Copy objects, directories, and buckets

AzCopy uses the [Put Block From URL](#) API, so data is copied directly between Google Cloud Storage and storage servers. These copy operations don't use the network bandwidth of your computer.

## TIP

The examples in this section enclose path arguments with single quotes (''). Use single quotes in all command shells except for the Windows Command Shell (cmd.exe). If you're using a Windows Command Shell (cmd.exe), enclose path arguments with double quotes ("") instead of single quotes ('').

These examples also work with accounts that have a hierarchical namespace. [Multi-protocol access on Data Lake Storage](#) enables you to use the same URL syntax (`blob.core.windows.net`) on those accounts.

## Copy an object

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

### Syntax

```
azcopy copy 'https://storage.cloud.google.com/<bucket-name>/<object-name>' 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<blob-name>'
```

### Example

```
azcopy copy 'https://storage.cloud.google.com/mybucket/myobject'
'https://mystorageaccount.blob.core.windows.net/mycontainer/myblob'
```

## Copy a directory

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

### Syntax

```
azcopy copy 'https://storage.cloud.google.com/<bucket-name>/<directory-name>' 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<directory-name>' --recursive=true
```

### Example

```
azcopy copy 'https://storage.cloud.google.com/mybucket/mydirectory'
'https://mystorageaccount.blob.core.windows.net/mycontainer/mydirectory' --recursive=true
```

## NOTE

This example appends the `--recursive` flag to copy files in all sub-directories.

## Copy the contents of a directory

You can copy the contents of a directory without copying the containing directory itself by using the wildcard symbol (\*).

### Syntax

```
azcopy copy 'https://storage.cloud.google.com/<bucket-name>/<directory-name>/*' 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<directory-name>' --recursive=true
```

### Example

```
azcopy copy 'https://storage.cloud.google.com/mybucket/mydirectory/*'
'https://mystorageaccount.blob.core.windows.net/mycontainer/mydirectory' --recursive=true
```

## Copy a Cloud Storage bucket

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

## Syntax

```
azcopy copy 'https://storage.cloud.google.com/<bucket-name>' 'https://<storage-account-name>.blob.core.windows.net' --recursive=true
```

## Example

```
azcopy copy 'https://storage.cloud.google.com/mybucket' 'https://mystorageaccount.blob.core.windows.net' --recursive=true
```

## Copy all buckets in a Google Cloud project

First, set the `GOOGLE_CLOUD_PROJECT` to project ID of Google Cloud project.

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

## Syntax

```
azcopy copy 'https://storage.cloud.google.com/' 'https://<storage-account-name>.blob.core.windows.net' --recursive=true
```

## Example

```
azcopy copy 'https://storage.cloud.google.com/' 'https://mystorageaccount.blob.core.windows.net' --recursive=true
```

## Copy a subset of buckets in a Google Cloud project

First, set the `GOOGLE_CLOUD_PROJECT` to project ID of Google Cloud project.

Copy a subset of buckets by using a wildcard symbol (\*) in the bucket name. Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

## Syntax

```
azcopy copy 'https://storage.cloud.google.com/<bucket*name>' 'https://<storage-account-name>.blob.core.windows.net' --recursive=true
```

## Example

```
azcopy copy 'https://storage.cloud.google.com/my*bucket' 'https://mystorageaccount.blob.core.windows.net' --recursive=true
```

## Handle differences in bucket naming rules

Google Cloud Storage has a different set of naming conventions for bucket names as compared to Azure blob containers. You can read about them [here](#). If you choose to copy a group of buckets to an Azure storage account, the copy operation might fail because of naming differences.

AzCopy handles three of the most common issues that can arise; buckets that contain periods, buckets that contain consecutive hyphens, and buckets that contain underscores. Google Cloud Storage bucket names can contain periods and consecutive hyphens, but a container in Azure can't. AzCopy replaces periods with hyphens and consecutive hyphens with a number that represents the number of consecutive hyphens (For example: a bucket named `my---bucket` becomes `my-4-bucket` . If the bucket name has an underscore (`_`), then AzCopy replaces the underscore with a hyphen. For example, a bucket named `my_bucket` becomes `my-bucket` .

## Handle differences in object naming rules

Google Cloud Storage has a different set of naming conventions for object names as compared to Azure blobs.

You can read about them [here](#).

Azure Storage does not permit object names (or any segment in the virtual directory path) to end with trailing dots (For example `my-bucket...`). Trailing dots are trimmed off when the copy operation is performed.

## Handle differences in object metadata

Google Cloud Storage and Azure allow different sets of characters in the names of object keys. You can read about metadata in Google Cloud Storage [here](#). On the Azure side, blob object keys adhere to the naming rules for [C# identifiers](#).

As part of an AzCopy `copy` command, you can provide a value for optional the `s2s-handle-invalid-metadata` flag that specifies how you would like to handle files where the metadata of the file contains incompatible key names. The following table describes each flag value.

FLAG VALUE	DESCRIPTION
<code>ExcludeIfInvalid</code>	(Default option) The metadata isn't included in the transferred object. AzCopy logs a warning.
<code>FailIfInvalid</code>	Objects aren't copied. AzCopy logs an error and includes that error in the failed count that appears in the transfer summary.
<code>RenameIfInvalid</code>	AzCopy resolves the invalid metadata key, and copies the object to Azure using the resolved metadata key value pair. To learn exactly what steps AzCopy takes to rename object keys, see the <a href="#">How AzCopy renames object keys</a> section below. If AzCopy is unable to rename the key, then the object won't be copied.

### How AzCopy renames object keys

AzCopy performs these steps:

1. Replaces invalid characters with '\_'.
2. Adds the string `rename_` to the beginning of a new valid key.

This key will be used to save the original metadata **value**.

3. Adds the string `rename_key_` to the beginning of a new valid key. This key will be used to save original metadata **invalid key**. You can use this key to try to recover the metadata in Azure side since metadata key is preserved as a value on the Blob storage service.

## Next steps

Find more examples in these articles:

- [Examples: Upload](#)
- [Examples: Download](#)
- [Examples: Copy between accounts](#)
- [Examples: Synchronize](#)
- [Examples: Amazon S3 buckets](#)
- [Examples: Azure Files](#)
- [Tutorial: Migrate on-premises data to cloud storage by using AzCopy](#)

See these articles to configure settings, optimize performance, and troubleshoot issues:

- [AzCopy configuration settings](#)
- [Optimize the performance of AzCopy](#)
- [Find errors and resume jobs by using log and plan files in AzCopy](#)
- [Troubleshoot problems with AzCopy v10](#)

# Copy and transform data in Azure Data Lake Storage Gen2 using Azure Data Factory or Azure Synapse Analytics

8/22/2022 • 33 minutes to read • [Edit Online](#)

APPLIES TO: Azure Data Factory Azure Synapse Analytics

Azure Data Lake Storage Gen2 (ADLS Gen2) is a set of capabilities dedicated to big data analytics built into [Azure Blob storage](#). You can use it to interface with your data by using both file system and object storage paradigms.

This article outlines how to use Copy Activity to copy data from and to Azure Data Lake Storage Gen2, and use Data Flow to transform data in Azure Data Lake Storage Gen2. To learn more, read the introductory article for [Azure Data Factory](#) or [Azure Synapse Analytics](#).

## TIP

For data lake or data warehouse migration scenario, learn more in [Migrate data from your data lake or data warehouse to Azure](#).

## Supported capabilities

This Azure Data Lake Storage Gen2 connector is supported for the following capabilities:

SUPPORTED CAPABILITIES	IR	MANAGED PRIVATE ENDPOINT
<a href="#">Copy activity</a> (source/sink)	① ②	✓
<a href="#">Mapping data flow</a> (source/sink)	①	✓
<a href="#">Lookup activity</a>	① ②	✓
<a href="#">GetMetadata activity</a>	① ②	✓
<a href="#">Delete activity</a>	① ②	✓

① Azure integration runtime ② Self-hosted integration runtime

For Copy activity, with this connector you can:

- Copy data from/to Azure Data Lake Storage Gen2 by using account key, service principal, or managed identities for Azure resources authentications.
- Copy files as-is or parse or generate files with [supported file formats and compression codecs](#).
- [Preserve file metadata during copy](#).
- [Preserve ACLs](#) when copying from Azure Data Lake Storage Gen1/Gen2.

## Get started

**TIP**

For a walk-through of how to use the Data Lake Storage Gen2 connector, see [Load data into Azure Data Lake Storage Gen2](#).

To perform the Copy activity with a pipeline, you can use one of the following tools or SDKs:

- [The Copy Data tool](#)
- [The Azure portal](#)
- [The .NET SDK](#)
- [The Python SDK](#)
- [Azure PowerShell](#)
- [The REST API](#)
- [The Azure Resource Manager template](#)

## Create an Azure Data Lake Storage Gen2 linked service using UI

Use the following steps to create an Azure Data Lake Storage Gen2 linked service in the Azure portal UI.

1. Browse to the Manage tab in your Azure Data Factory or Synapse workspace and select Linked Services, then click New:

- [Azure Data Factory](#)
- [Azure Synapse](#)

The screenshot shows the Azure Data Factory 'Linked services' page. On the left, there's a sidebar with various options like 'Connections', 'Source control', 'Author', and 'Security'. The 'Connections' section has 'Linked services' selected and highlighted with a red box. Below it, there are other options like 'Integration runtimes', 'Azure Purview (Preview)', 'Source control', 'Author', 'Triggers', 'Global parameters', 'Security', 'Customer managed key', 'Credentials', and 'Managed private endpoints'. To the right, the main area is titled 'Linked services' and contains a message: 'Linked service defines the connection information to a data store or compute. Learn more'. Below this is a '+ New' button, which is also highlighted with a red box. There's a 'Filter by name' input field and a table header with columns 'Name', 'Type', 'Related', and 'Annotations'. At the bottom, it says 'No linked service to show' and 'If you expected to see results, try changing your filters or create a new linked services.' A 'Create linked service' button is at the bottom right.

2. Search for Azure Data Lake Storage Gen2 and select the Azure Data Lake Storage Gen2 connector.

## New linked service

Data store   Compute

 Azure Data Lake Storage Gen2

All   Azure   Database   File   Generic protocol   NoSQL   Services and apps



Azure Data Lake Storage  
Gen2

Continue

Cancel

3. Configure the service details, test the connection, and create the new linked service.

## New linked service (Azure Data Lake Storage Gen2)

Name \*

Description

Connect via integration runtime \* ⓘ

Authentication method

Account selection method ⓘ

From Azure subscription  Enter manually

Azure subscription ⓘ

Storage account name \*

Test connection ⓘ

To linked service  To file path

Annotations

+ New

---

Create Back  Test connection Cancel

## Connector configuration details

The following sections provide information about properties that are used to define Data Factory and Synapse pipeline entities specific to Data Lake Storage Gen2.

## Linked service properties

The Azure Data Lake Storage Gen2 connector supports the following authentication types. See the corresponding sections for details:

- [Account key authentication](#)
- [Service principal authentication](#)
- [System-assigned managed identity authentication](#)
- [User-assigned managed identity authentication](#)

**NOTE**

- If want to use the public Azure integration runtime to connect to the Data Lake Storage Gen2 by leveraging the **Allow trusted Microsoft services to access this storage account** option enabled on Azure Storage firewall, you must use [managed identity authentication](#). For more information about the Azure Storage firewalls settings, see [Configure Azure Storage firewalls and virtual networks](#).
- When you use PolyBase or COPY statement to load data into Azure Synapse Analytics, if your source or staging Data Lake Storage Gen2 is configured with an Azure Virtual Network endpoint, you must use managed identity authentication as required by Azure Synapse. See the [managed identity authentication](#) section with more configuration prerequisites.

**Account key authentication**

To use storage account key authentication, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to <b>AzureBlobFS</b> .	Yes
url	Endpoint for Data Lake Storage Gen2 with the pattern of <code>https://&lt;accountname&gt;.dfs.core.windows.net</code>	Yes
accountKey	Account key for Data Lake Storage Gen2. Mark this field as a SecureString to store it securely, or <a href="#">reference a secret stored in Azure Key Vault</a> .	Yes
connectVia	The <a href="#">integration runtime</a> to be used to connect to the data store. You can use the Azure integration runtime or a self-hosted integration runtime if your data store is in a private network. If this property isn't specified, the default Azure integration runtime is used.	No

**NOTE**

Secondary ADLS file system endpoint is not supported when using account key authentication. You can use other authentication types.

**Example:**

```
{
 "name": "AzureDataLakeStorageGen2LinkedService",
 "properties": {
 "type": "AzureBlobFS",
 "typeProperties": {
 "url": "https://<accountname>.dfs.core.windows.net",
 "accountkey": {
 "type": "SecureString",
 "value": "<accountkey>"
 }
 },
 "connectVia": {
 "referenceName": "<name of Integration Runtime>",
 "type": "IntegrationRuntimeReference"
 }
 }
}
```

## Service principal authentication

To use service principal authentication, follow these steps.

1. Register an application entity in Azure Active Directory (Azure AD) by following the steps in [Register your application with an Azure AD tenant](#). Make note of the following values, which you use to define the linked service:
  - Application ID
  - Application key
  - Tenant ID
2. Grant the service principal proper permission. See examples on how permission works in Data Lake Storage Gen2 from [Access control lists on files and directories](#)
  - **As source:** In Storage Explorer, grant at least **Execute** permission for ALL upstream folders and the file system, along with **Read** permission for the files to copy. Alternatively, in Access control (IAM), grant at least the **Storage Blob Data Reader** role.
  - **As sink:** In Storage Explorer, grant at least **Execute** permission for ALL upstream folders and the file system, along with **Write** permission for the sink folder. Alternatively, in Access control (IAM), grant at least the **Storage Blob Data Contributor** role.

### NOTE

If you use UI to author and the service principal is not set with "Storage Blob Data Reader/Contributor" role in IAM, when doing test connection or browsing/navigating folders, choose "Test connection to file path" or "Browse from specified path", and specify a path with **Read + Execute** permission to continue.

These properties are supported for the linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to <b>AzureBlobFS</b> .	Yes
url	Endpoint for Data Lake Storage Gen2 with the pattern of <div style="border: 1px solid #ccc; padding: 2px;"><a href="https://&lt;accountname&gt;.dfs.core.windows.net">https://&lt;accountname&gt;.dfs.core.windows.net</a></div>	Yes

PROPERTY	DESCRIPTION	REQUIRED
servicePrincipalId	Specify the application's client ID.	Yes
servicePrincipalCredentialType	The credential type to use for service principal authentication. Allowed values are <b>ServicePrincipalKey</b> and <b>ServicePrincipalCert</b> .	Yes
servicePrincipalCredential	The service principal credential. When you use <b>ServicePrincipalKey</b> as the credential type, specify the application's key. Mark this field as <b>SecureString</b> to store it securely, or <a href="#">reference a secret stored in Azure Key Vault</a> . When you use <b>ServicePrincipalCert</b> as the credential, reference a certificate in Azure Key Vault, and ensure the certificate content type is <b>PKCS #12</b> .	Yes
servicePrincipalKey	Specify the application's key. Mark this field as <b>SecureString</b> to store it securely, or <a href="#">reference a secret stored in Azure Key Vault</a> . This property is still supported as-is for <code>servicePrincipalId + servicePrincipalKey</code> . As ADF adds new service principal certificate authentication, the new model for service principal authentication is <code>servicePrincipalId + servicePrincipalCredentialType + servicePrincipalCredential</code> .	No
tenant	Specify the tenant information (domain name or tenant ID) under which your application resides. Retrieve it by hovering the mouse in the upper-right corner of the Azure portal.	Yes
azureCloudType	For service principal authentication, specify the type of Azure cloud environment to which your Azure Active Directory application is registered. Allowed values are <b>AzurePublic</b> , <b>AzureChina</b> , <b>AzureUsGovernment</b> , and <b>AzureGermany</b> . By default, the data factory or Synapse pipeline's cloud environment is used.	No
connectVia	The <a href="#">integration runtime</a> to be used to connect to the data store. You can use the Azure integration runtime or a self-hosted integration runtime if your data store is in a private network. If not specified, the default Azure integration runtime is used.	No

## Example: using service principal key authentication

You can also store service principal key in Azure Key Vault.

```
{
 "name": "AzureDataLakeStorageGen2LinkedService",
 "properties": {
 "type": "AzureBlobFS",
 "typeProperties": {
 "url": "https://<accountname>.dfs.core.windows.net",
 "servicePrincipalId": "<service principal id>",
 "servicePrincipalCredentialType": "ServicePrincipalKey",
 "servicePrincipalCredential": {
 "type": "SecureString",
 "value": "<service principal key>"
 },
 "tenant": "<tenant info, e.g. microsoft.onmicrosoft.com>"
 },
 "connectVia": {
 "referenceName": "<name of Integration Runtime>",
 "type": "IntegrationRuntimeReference"
 }
 }
}
```

## Example: using service principal certificate authentication

```
{
 "name": "AzureDataLakeStorageGen2LinkedService",
 "properties": {
 "type": "AzureBlobFS",
 "typeProperties": {
 "url": "https://<accountname>.dfs.core.windows.net",
 "servicePrincipalId": "<service principal id>",
 "servicePrincipalCredentialType": "ServicePrincipalCert",
 "servicePrincipalCredential": {
 "type": "AzureKeyVaultSecret",
 "store": {
 "referenceName": "<AKV reference>",
 "type": "LinkedServiceReference"
 },
 "secretName": "<certificate name in AKV>"
 },
 "tenant": "<tenant info, e.g. microsoft.onmicrosoft.com>"
 },
 "connectVia": {
 "referenceName": "<name of Integration Runtime>",
 "type": "IntegrationRuntimeReference"
 }
 }
}
```

## System-assigned managed identity authentication

A data factory or Synapse workspace can be associated with a [system-assigned managed identity](#). You can directly use this system-assigned managed identity for Data Lake Storage Gen2 authentication, similar to using your own service principal. It allows this designated factory or workspace to access and copy data to or from your Data Lake Storage Gen2.

To use system-assigned managed identity authentication, follow these steps.

1. [Retrieve the system-assigned managed identity information](#) by copying the value of the **managed identity object ID** generated along with your data factory or Synapse workspace.

2. Grant the system-assigned managed identity proper permission. See examples on how permission works in Data Lake Storage Gen2 from [Access control lists on files and directories](#).

- **As source:** In Storage Explorer, grant at least **Execute** permission for ALL upstream folders and the file system, along with **Read** permission for the files to copy. Alternatively, in Access control (IAM), grant at least the **Storage Blob Data Reader** role.
- **As sink:** In Storage Explorer, grant at least **Execute** permission for ALL upstream folders and the file system, along with **Write** permission for the sink folder. Alternatively, in Access control (IAM), grant at least the **Storage Blob Data Contributor** role.

These properties are supported for the linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to <b>AzureBlobFS</b> .	Yes
url	Endpoint for Data Lake Storage Gen2 with the pattern of <code>https://&lt;accountname&gt;.dfs.core.windows.net</code>	Yes
connectVia	The <a href="#">integration runtime</a> to be used to connect to the data store. You can use the Azure integration runtime or a self-hosted integration runtime if your data store is in a private network. If not specified, the default Azure integration runtime is used.	No

**Example:**

```
{
 "name": "AzureDataLakeStorageGen2LinkedService",
 "properties": {
 "type": "AzureBlobFS",
 "typeProperties": {
 "url": "https://<accountname>.dfs.core.windows.net",
 },
 "connectVia": {
 "referenceName": "<name of Integration Runtime>",
 "type": "IntegrationRuntimeReference"
 }
 }
}
```

### User-assigned managed identity authentication

A data factory can be assigned with one or multiple [user-assigned managed identities](#). You can use this user-assigned managed identity for Blob storage authentication, which allows to access and copy data from or to Data Lake Storage Gen2. To learn more about managed identities for Azure resources, see [Managed identities for Azure resources](#)

To use user-assigned managed identity authentication, follow these steps:

1. [Create one or multiple user-assigned managed identities](#) and grant access to Azure Data Lake Storage Gen2. See examples on how permission works in Data Lake Storage Gen2 from [Access control lists on files and directories](#).

- **As source:** In Storage Explorer, grant at least **Execute** permission for ALL upstream folders and the file system, along with **Read** permission for the files to copy. Alternatively, in Access control (IAM), grant at least the **Storage Blob Data Reader** role.
- **As sink:** In Storage Explorer, grant at least **Execute** permission for ALL upstream folders and the file system, along with **Write** permission for the sink folder. Alternatively, in Access control (IAM), grant at least the **Storage Blob Data Contributor** role.

2. Assign one or multiple user-assigned managed identities to your data factory and [create credentials](#) for each user-assigned managed identity.

These properties are supported for the linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to <b>AzureBlobFS</b> .	Yes
url	Endpoint for Data Lake Storage Gen2 with the pattern of <code>https://&lt;accountname&gt;.dfs.core.windows.net</code>	Yes
credentials	Specify the user-assigned managed identity as the credential object.	Yes
connectVia	The <a href="#">integration runtime</a> to be used to connect to the data store. You can use the Azure integration runtime or a self-hosted integration runtime if your data store is in a private network. If not specified, the default Azure integration runtime is used.	No

**Example:**

```
{
 "name": "AzureDataLakeStorageGen2LinkedService",
 "properties": {
 "type": "AzureBlobFS",
 "typeProperties": {
 "url": "https://<accountname>.dfs.core.windows.net",
 "credential": {
 "referenceName": "credential1",
 "type": "CredentialReference"
 }
 },
 "connectVia": {
 "referenceName": "<name of Integration Runtime>",
 "type": "IntegrationRuntimeReference"
 }
 }
}
```

#### NOTE

If you use Data Factory UI to author and the managed identity is not set with "Storage Blob Data Reader/Contributor" role in IAM, when doing test connection or browsing/navigating folders, choose "Test connection to file path" or "Browse from specified path", and specify a path with **Read + Execute** permission to continue.

## IMPORTANT

If you use PolyBase or COPY statement to load data from Data Lake Storage Gen2 into Azure Synapse Analytics, when you use managed identity authentication for Data Lake Storage Gen2, make sure you also follow steps 1 to 3 in [this guidance](#). Those steps will register your server with Azure AD and assign the Storage Blob Data Contributor role to your server. Data Factory handles the rest. If you configure Blob storage with an Azure Virtual Network endpoint, you also need to have **Allow trusted Microsoft services to access this storage account** turned on under Azure Storage account **Firewalls and Virtual networks** settings menu as required by Azure Synapse.

## Dataset properties

For a full list of sections and properties available for defining datasets, see [Datasets](#).

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- [Avro format](#)
- [Binary format](#)
- [Delimited text format](#)
- [Excel format](#)
- [JSON format](#)
- [ORC format](#)
- [Parquet format](#)
- [XML format](#)

The following properties are supported for Data Lake Storage Gen2 under `location` settings in the format-based dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under <code>location</code> in the dataset must be set to <b>AzureBlobFSLocation</b> .	Yes
fileSystem	The Data Lake Storage Gen2 file system name.	No
folderPath	The path to a folder under the given file system. If you want to use a wildcard to filter folders, skip this setting and specify it in activity source settings.	No
fileName	The file name under the given <code>fileSystem + folderPath</code> . If you want to use a wildcard to filter files, skip this setting and specify it in activity source settings.	No

**Example:**

```
{
 "name": "DelimitedTextDataset",
 "properties": {
 "type": "DelimitedText",
 "linkedServiceName": {
 "referenceName": "<Data Lake Storage Gen2 linked service name>",
 "type": "LinkedServiceReference"
 },
 "schema": [< physical schema, optional, auto retrieved during authoring >],
 "typeProperties": {
 "location": {
 "type": "AzureBlobFSLocation",
 "fileSystem": "filesystemname",
 "folderPath": "folder/subfolder"
 },
 "columnDelimiter": ",",
 "quoteChar": "\"",
 "firstRowAsHeader": true,
 "compressionCodec": "gzip"
 }
 }
}
```

## Copy activity properties

For a full list of sections and properties available for defining activities, see [Copy activity configurations](#) and [Pipelines and activities](#). This section provides a list of properties supported by the Data Lake Storage Gen2 source and sink.

### Azure Data Lake Storage Gen2 as a source type

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- [Avro format](#)
- [Binary format](#)
- [Delimited text format](#)
- [Excel format](#)
- [JSON format](#)
- [ORC format](#)
- [Parquet format](#)
- [XML format](#)

You have several options to copy data from ADLS Gen2:

- Copy from the given path specified in the dataset.
- Wildcard filter against folder path or file name, see `wildcardFolderPath` and `wildcardFileName`.
- Copy the files defined in a given text file as file set, see `fileListPath`.

The following properties are supported for Data Lake Storage Gen2 under `storeSettings` settings in format-based copy source:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under <code>storeSettings</code> must be set to <code>AzureBlobFSReadSettings</code> .	Yes

PROPERTY	DESCRIPTION	REQUIRED
<b><i>Locate the files to copy:</i></b>		
OPTION 1: static path - wildcardFolderPath	Copy from the given file system or folder/file path specified in the dataset. If you want to copy all files from a file system/folder, additionally specify <code>wildcardFileName</code> as <code>*</code> .	
OPTION 2: wildcard - wildcardFolderPath	The folder path with wildcard characters under the given file system configured in dataset to filter source folders. Allowed wildcards are: <code>*</code> (matches zero or more characters) and <code>?</code> (matches zero or single character); use <code>^</code> to escape if your actual folder name has wildcard or this escape char inside. See more examples in <a href="#">Folder and file filter examples</a> .	No
OPTION 2: wildcard - wildcardFileName	The file name with wildcard characters under the given file system + folderPath/wildcardFolderPath to filter source files. Allowed wildcards are: <code>*</code> (matches zero or more characters) and <code>?</code> (matches zero or single character); use <code>^</code> to escape if your actual file name has wildcard or this escape char inside. See more examples in <a href="#">Folder and file filter examples</a> .	Yes
OPTION 3: a list of files - fileListPath	Indicates to copy a given file set. Point to a text file that includes a list of files you want to copy, one file per line, which is the relative path to the path configured in the dataset. When using this option, do not specify file name in dataset. See more examples in <a href="#">File list examples</a> .	No
<b><i>Additional settings:</i></b>		
recursive	Indicates whether the data is read recursively from the subfolders or only from the specified folder. Note that when recursive is set to true and the sink is a file-based store, an empty folder or subfolder isn't copied or created at the sink. Allowed values are <b>true</b> (default) and <b>false</b> . This property doesn't apply when you configure <code>fileListPath</code> .	No

PROPERTY	DESCRIPTION	REQUIRED
deleteFilesAfterCompletion	<p>Indicates whether the binary files will be deleted from source store after successfully moving to the destination store. The file deletion is per file, so when copy activity fails, you will see some files have already been copied to the destination and deleted from source, while others are still remaining on source store.</p> <p>This property is only valid in binary files copy scenario. The default value: false.</p>	No
modifiedDatetimeStart	<p>Files filter based on the attribute: Last Modified.</p> <p>The files will be selected if their last modified time is greater than or equal to <code>modifiedDatetimeStart</code> and less than <code>modifiedDatetimeEnd</code>. The time is applied to UTC time zone in the format of "2018-12-01T05:00:00Z".</p> <p>The properties can be NULL, which means no file attribute filter will be applied to the dataset. When <code>modifiedDatetimeStart</code> has datetime value but <code>modifiedDatetimeEnd</code> is NULL, it means the files whose last modified attribute is greater than or equal with the datetime value will be selected.</p> <p>When <code>modifiedDatetimeEnd</code> has datetime value but <code>modifiedDatetimeStart</code> is NULL, it means the files whose last modified attribute is less than the datetime value will be selected.</p> <p>This property doesn't apply when you configure <code>fileListPath</code>.</p>	No
modifiedDatetimeEnd	Same as above.	No
enablePartitionDiscovery	<p>For files that are partitioned, specify whether to parse the partitions from the file path and add them as additional source columns.</p> <p>Allowed values are <b>false</b> (default) and <b>true</b>.</p>	No

PROPERTY	DESCRIPTION	REQUIRED
partitionRootPath	<p>When partition discovery is enabled, specify the absolute root path in order to read partitioned folders as data columns.</p> <p>If it is not specified, by default,</p> <ul style="list-style-type: none"> <li>- When you use file path in dataset or list of files on source, partition root path is the path configured in dataset.</li> <li>- When you use wildcard folder filter, partition root path is the sub-path before the first wildcard.</li> </ul> <p>For example, assuming you configure the path in dataset as "root/folder/year=2020/month=08/day=27":</p> <ul style="list-style-type: none"> <li>- If you specify partition root path as "root/folder/year=2020", copy activity will generate two more columns <code>month</code> and <code>day</code> with value "08" and "27" respectively, in addition to the columns inside the files.</li> <li>- If partition root path is not specified, no extra column will be generated.</li> </ul>	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

**Example:**

```

"activities": [
 {
 "name": "CopyFromADLSGen2",
 "type": "Copy",
 "inputs": [
 {
 "referenceName": "<Delimited text input dataset name>",
 "type": "DatasetReference"
 }
],
 "outputs": [
 {
 "referenceName": "<output dataset name>",
 "type": "DatasetReference"
 }
],
 "typeProperties": {
 "source": {
 "type": "DelimitedTextSource",
 "formatSettings": {
 "type": "DelimitedTextReadSettings",
 "skipLineCount": 10
 },
 "storeSettings": {
 "type": "AzureBlobFSReadSettings",
 "recursive": true,
 "wildcardFolderPath": "myfolder*A",
 "wildcardFileName": "*.csv"
 }
 },
 "sink": {
 "type": "<sink type>"
 }
 }
 }
]

```

## Azure Data Lake Storage Gen2 as a sink type

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- [Avro format](#)
- [Binary format](#)
- [Delimited text format](#)
- [JSON format](#)
- [ORC format](#)
- [Parquet format](#)

The following properties are supported for Data Lake Storage Gen2 under `storeSettings` settings in format-based copy sink:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under <code>storeSettings</code> must be set to <code>AzureBlobFSWriteSettings</code> .	Yes

PROPERTY	DESCRIPTION	REQUIRED
copyBehavior	<p>Defines the copy behavior when the source is files from a file-based data store.</p> <p>Allowed values are:</p> <ul style="list-style-type: none"> <li>- <b>PreserveHierarchy (default):</b> Preserves the file hierarchy in the target folder. The relative path of the source file to the source folder is identical to the relative path of the target file to the target folder.</li> <li>- <b>FlattenHierarchy:</b> All files from the source folder are in the first level of the target folder. The target files have autogenerated names.</li> <li>- <b>MergeFiles:</b> Merges all files from the source folder to one file. If the file name is specified, the merged file name is the specified name. Otherwise, it's an autogenerated file name.</li> </ul>	No
blockSizeInMB	<p>Specify the block size in MB used to write data to ADLS Gen2. Learn more <a href="#">about Block Blobs</a>.</p> <p>Allowed value is <b>between 4 MB and 100 MB</b>.</p> <p>By default, ADF automatically determines the block size based on your source store type and data. For non-binary copy into ADLS Gen2, the default block size is 100 MB so as to fit in at most approximately 4.75-TB data. It may be not optimal when your data is not large, especially when you use Self-hosted Integration Runtime with poor network resulting in operation timeout or performance issue. You can explicitly specify a block size, while ensure <code>blockSizeInMB*50000</code> is big enough to store the data, otherwise copy activity run will fail.</p>	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

PROPERTY	DESCRIPTION	REQUIRED
metadata	<p>Set custom metadata when copy to sink. Each object under the <code>metadata</code> array represents an extra column. The <code>name</code> defines the metadata key name, and the <code>value</code> indicates the data value of that key. If <a href="#">preserve attributes feature</a> is used, the specified metadata will union/overwrite with the source file metadata.</p> <p>Allowed data values are:</p> <ul style="list-style-type: none"> <li>- <code>\$\$LASTMODIFIED</code> : a reserved variable indicates to store the source files' last modified time. Apply to file-based source with binary format only.</li> <li>- <a href="#">Expression</a></li> <li>- <a href="#">Static value</a></li> </ul>	No

Example:

```

"activities": [
 {
 "name": "CopyToADLSSGen2",
 "type": "Copy",
 "inputs": [
 {
 "referenceName": "<input dataset name>",
 "type": "DatasetReference"
 }
],
 "outputs": [
 {
 "referenceName": "<Parquet output dataset name>",
 "type": "DatasetReference"
 }
],
 "typeProperties": {
 "source": {
 "type": "<source type>"
 },
 "sink": {
 "type": "ParquetSink",
 "storeSettings": {
 "type": "AzureBlobFSWriteSettings",
 "copyBehavior": "PreserveHierarchy",
 "metadata": [
 {
 "name": "testKey1",
 "value": "value1"
 },
 {
 "name": "testKey2",
 "value": "value2"
 },
 {
 "name": "lastModifiedKey",
 "value": "$$LASTMODIFIED"
 }
]
 }
 }
 }
 }
]

```

## Folder and file filter examples

This section describes the resulting behavior of the folder path and file name with wildcard filters.

FOLDERPATH	FILENAME	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
Folder*	(Empty, use default)	false	FolderA <b>File1.csv</b> <b>File2.json</b> Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv

FOLDERPATH	FILENAME	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
Folder*	(Empty, use default)	true	FolderA <b>File1.csv</b> <b>File2.json</b> Subfolder1 <b>File3.csv</b> <b>File4.json</b> <b>File5.csv</b> AnotherFolderB <b>File6.csv</b>
Folder*	*.csv	false	FolderA <b>File1.csv</b> <b>File2.json</b> Subfolder1 File3.csv File4.json <b>File5.csv</b> AnotherFolderB <b>File6.csv</b>
Folder*	*.csv	true	FolderA <b>File1.csv</b> <b>File2.json</b> Subfolder1 <b>File3.csv</b> File4.json <b>File5.csv</b> AnotherFolderB <b>File6.csv</b>

## File list examples

This section describes the resulting behavior of using file list path in copy activity source.

Assuming you have the following source folder structure and want to copy the files in bold:

SAMPLE SOURCE STRUCTURE	CONTENT IN FILELISTTOCOPY.TXT	ADF CONFIGURATION
filesystem FolderA <b>File1.csv</b> File2.json Subfolder1 <b>File3.csv</b> File4.json <b>File5.csv</b> Metadata FileListToCopy.txt	File1.csv Subfolder1/File3.csv Subfolder1/File5.csv	<p><b>In dataset:</b></p> <ul style="list-style-type: none"> <li>- File system: <code>filesystem</code></li> <li>- Folder path: <code>FolderA</code></li> </ul> <p><b>In copy activity source:</b></p> <ul style="list-style-type: none"> <li>- File list path: <code>filesystem/Metadata/FileListToCopy.txt</code></li> </ul> <p>The file list path points to a text file in the same data store that includes a list of files you want to copy, one file per line with the relative path to the path configured in the dataset.</p>

## Some recursive and copyBehavior examples

This section describes the resulting behavior of the copy operation for different combinations of recursive and copyBehavior values.

RECURSIVE	COPYBEHAVIOR	SOURCE FOLDER STRUCTURE	RESULTING TARGET
true	preserveHierarchy	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target Folder1 is created with the same structure as the source:  Folder1 File1 File2 Subfolder1 File3 File4 File5
true	flattenHierarchy	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target Folder1 is created with the following structure:  Folder1 autogenerated name for File1 autogenerated name for File2 autogenerated name for File3 autogenerated name for File4 autogenerated name for File5
true	mergeFiles	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target Folder1 is created with the following structure:  Folder1 File1 + File2 + File3 + File4 + File5 contents are merged into one file with an autogenerated file name.
false	preserveHierarchy	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target Folder1 is created with the following structure:  Folder1 File1 File2  Subfolder1 with File3, File4, and File5 isn't picked up.

Recursive	CopyBehavior	Source Folder Structure	Resulting Target
false	flattenHierarchy	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target Folder1 is created with the following structure:  Folder1 autogenerated name for File1 autogenerated name for File2  Subfolder1 with File3, File4, and File5 isn't picked up.
false	mergeFiles	Folder1 File1 File2 Subfolder1 File3 File4 File5	The target Folder1 is created with the following structure:  Folder1 File1 + File2 contents are merged into one file with an autogenerated file name. autogenerated name for File1  Subfolder1 with File3, File4, and File5 isn't picked up.

## Preserve metadata during copy

When you copy files from Amazon S3/Azure Blob/Azure Data Lake Storage Gen2 to Azure Data Lake Storage Gen2/Azure Blob, you can choose to preserve the file metadata along with data. Learn more from [Preserve metadata](#).

## Preserve ACLs from Data Lake Storage Gen1/Gen2

When you copy files from Azure Data Lake Storage Gen1/Gen2 to Gen2, you can choose to preserve the POSIX access control lists (ACLs) along with data. Learn more from [Preserve ACLs from Data Lake Storage Gen1/Gen2 to Gen2](#).

### TIP

To copy data from Azure Data Lake Storage Gen1 into Gen2 in general, see [Copy data from Azure Data Lake Storage Gen1 to Gen2](#) for a walk-through and best practices.

## Mapping data flow properties

When you're transforming data in mapping data flows, you can read and write files from Azure Data Lake Storage Gen2 in the following formats:

- [Avro](#)
- [Common Data Model](#)
- [Delimited text](#)
- [Delta](#)

- Excel
- JSON
- Parquet

Format specific settings are located in the documentation for that format. For more information, see [Source transformation in mapping data flow](#) and [Sink transformation in mapping data flow](#).

## Source transformation

In the source transformation, you can read from a container, folder, or individual file in Azure Data Lake Storage Gen2. The **Source options** tab lets you manage how the files get read.

The screenshot shows the 'Source options' tab selected in the top navigation bar. Below it, there are several configuration fields:

- Wildcard paths:** A text input field containing "mycontainer / partdata/\*\*/\*.csv". To its right are three icons: a help icon, a plus sign for adding more patterns, and a trash bin for deleting.
- Partition root path:** A text input field containing "partdata". To its right is a help icon.
- List of files:** A checkbox followed by a help icon.
- Multiline rows:** A checkbox followed by a help icon.
- Column to store file name:** A text input field containing "fileName". To its right is a help icon.
- After completion \***: Three radio button options: "No action" (selected), "Delete source files", and "Move".
- Start time (UTC)**: A text input field containing "02/01/2020 00:00:00".
- End time (UTC)**: A text input field containing "02/02/2020 00:00:00".
- Filter by last modified**: A text input field.

**Wildcard path:** Using a wildcard pattern will instruct ADF to loop through each matching folder and file in a single Source transformation. This is an effective way to process multiple files within a single flow. Add multiple wildcard matching patterns with the + sign that appears when hovering over your existing wildcard pattern.

From your source container, choose a series of files that match a pattern. Only container can be specified in the dataset. Your wildcard path must therefore also include your folder path from the root folder.

Wildcard examples:

- `*` Represents any set of characters
- `**` Represents recursive directory nesting
- `?` Replaces one character
- `[]` Matches one of more characters in the brackets
- `/data/sales/**/*.csv` Gets all csv files under /data/sales
- `/data/sales/20??/**/` Gets all files in the 20th century
- `/data/sales/*/*/*.csv` Gets csv files two levels under /data/sales
- `/data/sales/2004/*/12/[XY]1?.csv` Gets all csv files in 2004 in December starting with X or Y prefixed by a two-digit number

**Partition Root Path:** If you have partitioned folders in your file source with a `key=value` format (for example, `year=2019`), then you can assign the top level of that partition folder tree to a column name in your data flow data stream.

First, set a wildcard to include all paths that are the partitioned folders plus the leaf files that you wish to read.

Wildcard paths mycontainer / partdata/\*\*/\*.\*csv + -

Partition root path partdata + -

List of files + -

Multiline rows + -

Column to store file name myfile + -

After completion \*  No action  Delete source files  Move

Use the Partition Root Path setting to define what the top level of the folder structure is. When you view the contents of your data via a data preview, you'll see that ADF will add the resolved partitions found in each of your folder levels.

Projection	Optimize	Inspect	Data Preview	Description
00	* UPDATE 0	X DELETE 0	+ UPSERT 0	SEARCH LOOKUP 0
Rating abc	Rotten Tomato abc	releaseyear 123	Month 123	myfile abc
1	54	2019	7	/partdata/releaseyear=2019/...
7	80	2019	7	/partdata/releaseyear=2019/...
6	92	2019	7	/partdata/releaseyear=2019/...
4	82	2019	7	/partdata/releaseyear=2019/...
9	92	2019	7	/partdata/releaseyear=2019/...
4	59	2019	7	/partdata/releaseyear=2019/...
3	83	2019	7	/partdata/releaseyear=2019/...
2	63	2019	7	/partdata/releaseyear=2019/...
4	63	2019	7	/partdata/releaseyear=2019/...
8	50	2019	7	/partdata/releaseyear=2019/...

**List of files:** This is a file set. Create a text file that includes a list of relative path files to process. Point to this text file.

**Column to store file name:** Store the name of the source file in a column in your data. Enter a new column name here to store the file name string.

**After completion:** Choose to do nothing with the source file after the data flow runs, delete the source file, or move the source file. The paths for the move are relative.

To move source files to another location post-processing, first select "Move" for file operation. Then, set the "from" directory. If you're not using any wildcards for your path, then the "from" setting will be the same folder as your source folder.

If you have a source path with wildcard, your syntax will look like this below:

```
/data/sales/20??/**/*.*csv
```

You can specify "from" as

```
/data/sales
```

And "to" as

```
/backup/priorSales
```

In this case, all files that were sourced under /data/sales are moved to /backup/priorSales.

#### NOTE

File operations run only when you start the data flow from a pipeline run (a pipeline debug or execution run) that uses the Execute Data Flow activity in a pipeline. File operations *do not* run in Data Flow debug mode.

**Filter by last modified:** You can filter which files you process by specifying a date range of when they were last modified. All date-times are in UTC.

**Enable change data capture:** If true, you will get new or changed files only from the last run. Initial load of full snapshot data will always be gotten in the first run, followed by capturing new or changed files only in next runs. For more details, see [Change data capture](#).

The screenshot shows the 'Source options' tab of the Azure Data Flow designer. It includes fields for Wildcard paths, Partition root path, Allow no files found, List of files, Multiline rows, and a checkbox for Enable change data capture. The 'Enable change data capture' checkbox is highlighted with a red box. Other tabs like Source settings, Projection, Optimize, Inspect, and Data preview are also visible.

#### Sink properties

In the sink transformation, you can write to either a container or folder in Azure Data Lake Storage Gen2. the **Settings** tab lets you manage how the files get written.

The screenshot shows the 'Settings' tab of the Azure Data Flow designer. It includes options for Clear the folder (checked), File name option (Default selected), Pattern, Per partition, As data in column, and Output to single file. The 'Clear the folder' checkbox is checked.

**Clear the folder:** Determines whether or not the destination folder gets cleared before the data is written.

**File name option:** Determines how the destination files are named in the destination folder. The file name options are:

- **Default:** Allow Spark to name files based on PART defaults.
- **Pattern:** Enter a pattern that enumerates your output files per partition. For example, `loans[n].csv` will create `loans1.csv`, `loans2.csv`, and so on.
- **Per partition:** Enter one file name per partition.
- **As data in column:** Set the output file to the value of a column. The path is relative to the dataset container, not the destination folder. If you have a folder path in your dataset, it will be overridden.
- **Output to a single file:** Combine the partitioned output files into a single named file. The path is relative to the dataset folder. Please be aware that the merge operation can possibly fail based upon node size. This

option is not recommended for large datasets.

**Quote all:** Determines whether to enclose all values in quotes

#### umask

You can optionally set the `umask` for files using POSIX read, write, execute flags for owner, user and group.

### Pre-processing and post-processing commands

You can optionally execute Hadoop filesystem commands before or after writing to an ADLS Gen2 sink. The following commands are supported:

- `cp`
- `mv`
- `rm`
- `mkdir`

Examples:

- `mkdir /folder1`
- `mkdir -p folder1`
- `mv /folder1/*.* /folder2/`
- `cp /folder1/file1.txt /folder2`
- `rm -r /folder1`

Parameters are also supported through expression builder, for example:

```
mkdir -p ${tempPath}/commands/c1/c2 mv ${tempPath}/commands/*.* ${tempPath}/commands/c1/c2
```

By default, folders are created as user/root. Refer to the top level container with '/'.

## Lookup activity properties

To learn details about the properties, check [Lookup activity](#).

## GetMetadata activity properties

To learn details about the properties, check [GetMetadata activity](#)

## Delete activity properties

To learn details about the properties, check [Delete activity](#)

## Legacy models

#### NOTE

The following models are still supported as-is for backward compatibility. You are suggested to use the new model mentioned in above sections going forward, and the ADF authoring UI has switched to generating the new model.

### Legacy dataset model

PROPERTY	DESCRIPTION	REQUIRED
----------	-------------	----------

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the dataset must be set to <b>AzureBlobFSFile</b> .	Yes
folderPath	<p>Path to the folder in Data Lake Storage Gen2. If not specified, it points to the root.</p> <p>Wildcard filter is supported. Allowed wildcards are <code>*</code> (matches zero or more characters) and <code>?</code> (matches zero or single character). Use <code>^</code> to escape if your actual folder name has a wildcard or this escape char is inside.</p> <p>Examples: filesystem/folder/. See more examples in <a href="#">Folder and file filter examples</a>.</p>	No
fileName	<p>Name or wildcard filter for the files under the specified "folderPath". If you don't specify a value for this property, the dataset points to all files in the folder.</p> <p>For filter, the wildcards allowed are <code>*</code> (matches zero or more characters) and <code>?</code> (matches zero or single character).</p> <ul style="list-style-type: none"> <li>- Example 1: <code>"fileName": "*.csv"</code></li> <li>- Example 2: <code>"fileName": "???20180427.txt"</code></li> </ul> <p>Use <code>^</code> to escape if your actual file name has a wildcard or this escape char is inside.</p> <p>When fileName isn't specified for an output dataset and <b>preserveHierarchy</b> isn't specified in the activity sink, the copy activity automatically generates the file name with the following pattern: <i>Data.[activity run ID GUID].[GUID if FlattenHierarchy].[format if configured].[compression if configured]</i>, for example, "Data.0a405f8a-93ff-4c6fb3be-f69616f1df7a.txt.gz". If you copy from a tabular source using a table name instead of a query, the name pattern is <i>[table name].[format].[compression if configured]</i>, for example, "MyTable.csv".</p>	No

PROPERTY	DESCRIPTION	REQUIRED
modifiedDatetimeStart	<p>Files filter based on the attribute Last Modified. The files are selected if their last modified time is greater than or equal to <code>modifiedDatetimeStart</code> and less than <code>modifiedDatetimeEnd</code>. The time is applied to the UTC time zone in the format of "2018-12-01T05:00:00Z".</p> <p>The overall performance of data movement is affected by enabling this setting when you want to do file filter with huge amounts of files.</p> <p>The properties can be NULL, which means no file attribute filter is applied to the dataset. When <code>modifiedDatetimeStart</code> has a datetime value but <code>modifiedDatetimeEnd</code> is NULL, it means the files whose last modified attribute is greater than or equal to the datetime value are selected. When <code>modifiedDatetimeEnd</code> has a datetime value but <code>modifiedDatetimeStart</code> is NULL, it means the files whose last modified attribute is less than the datetime value are selected.</p>	No
modifiedDatetimeEnd	<p>Files filter based on the attribute Last Modified. The files are selected if their last modified time is greater than or equal to <code>modifiedDatetimeStart</code> and less than <code>modifiedDatetimeEnd</code>. The time is applied to the UTC time zone in the format of "2018-12-01T05:00:00Z".</p> <p>The overall performance of data movement is affected by enabling this setting when you want to do file filter with huge amounts of files.</p> <p>The properties can be NULL, which means no file attribute filter is applied to the dataset. When <code>modifiedDatetimeStart</code> has a datetime value but <code>modifiedDatetimeEnd</code> is NULL, it means the files whose last modified attribute is greater than or equal to the datetime value are selected. When <code>modifiedDatetimeEnd</code> has a datetime value but <code>modifiedDatetimeStart</code> is NULL, it means the files whose last modified attribute is less than the datetime value are selected.</p>	No

PROPERTY	DESCRIPTION	REQUIRED
format	<p>If you want to copy files as is between file-based stores (binary copy), skip the format section in both the input and output dataset definitions.</p> <p>If you want to parse or generate files with a specific format, the following file format types are supported: <a href="#">TextFormat</a>, <a href="#">JsonFormat</a>, <a href="#">AvroFormat</a>, <a href="#">OrcFormat</a>, and <a href="#">ParquetFormat</a>. Set the <b>type</b> property under <b>format</b> to one of these values. For more information, see the <a href="#">Text format</a>, <a href="#">JSON format</a>, <a href="#">Avro format</a>, <a href="#">ORC format</a>, and <a href="#">Parquet format</a> sections.</p>	No (only for binary copy scenario)
compression	<p>Specify the type and level of compression for the data. For more information, see <a href="#">Supported file formats and compression codecs</a>.</p> <p>Supported types are  <span style="border: 1px solid black; padding: 2px;">**GZip**, **Deflate**, **BZip2**, and **ZipDeflate**</span></p> <p>Supported levels are <b>Optimal</b> and <b>Fastest</b>.</p>	No

#### TIP

To copy all files under a folder, specify **FolderPath** only.

To copy a single file with a given name, specify **FolderPath** with a folder part and **fileName** with a file name.

To copy a subset of files under a folder, specify **FolderPath** with a folder part and **fileName** with a wildcard filter.

**Example:**

```
{
 "name": "ADLSGen2Dataset",
 "properties": {
 "type": "AzureBlobFSFile",
 "linkedServiceName": {
 "referenceName": "<Azure Data Lake Storage Gen2 linked service name>",
 "type": "LinkedServiceReference"
 },
 "typeProperties": {
 "folderPath": "myfilesystem/myfolder",
 "fileName": "*",
 "modifiedDatetimeStart": "2018-12-01T05:00:00Z",
 "modifiedDatetimeEnd": "2018-12-01T06:00:00Z",
 "format": {
 "type": "TextFormat",
 "columnDelimiter": ",",
 "rowDelimiter": "\n"
 },
 "compression": {
 "type": "GZip",
 "level": "Optimal"
 }
 }
 }
}
```

## Legacy copy activity source model

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to <b>AzureBlobFSSource</b> .	Yes
recursive	Indicates whether the data is read recursively from the subfolders or only from the specified folder. When recursive is set to true and the sink is a file-based store, an empty folder or subfolder isn't copied or created at the sink. Allowed values are <b>true</b> (default) and <b>false</b> .	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

Example:

```

"activities": [
 {
 "name": "CopyFromADLSGen2",
 "type": "Copy",
 "inputs": [
 {
 "referenceName": "<ADLS Gen2 input dataset name>",
 "type": "DatasetReference"
 }
],
 "outputs": [
 {
 "referenceName": "<output dataset name>",
 "type": "DatasetReference"
 }
],
 "typeProperties": {
 "source": {
 "type": "AzureBlobFSSource",
 "recursive": true
 },
 "sink": {
 "type": "<sink type>"
 }
 }
 }
]

```

## Legacy copy activity sink model

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity sink must be set to <b>AzureBlobFSSink</b> .	Yes
copyBehavior	<p>Defines the copy behavior when the source is files from a file-based data store.</p> <p>Allowed values are:</p> <ul style="list-style-type: none"> <li>- <b>PreserveHierarchy</b> (default): Preserves the file hierarchy in the target folder. The relative path of the source file to the source folder is identical to the relative path of the target file to the target folder.</li> <li>- <b>FlattenHierarchy</b>: All files from the source folder are in the first level of the target folder. The target files have autogenerated names.</li> <li>- <b>MergeFiles</b>: Merges all files from the source folder to one file. If the file name is specified, the merged file name is the specified name. Otherwise, it's an autogenerated file name.</li> </ul>	No
maxConcurrentConnections	The upper limit of concurrent connections established to the data store during the activity run. Specify a value only when you want to limit concurrent connections.	No

## Example:

```
"activities": [
 {
 "name": "CopyToADLSGen2",
 "type": "Copy",
 "inputs": [
 {
 "referenceName": "<input dataset name>",
 "type": "DatasetReference"
 }
],
 "outputs": [
 {
 "referenceName": "<ADLS Gen2 output dataset name>",
 "type": "DatasetReference"
 }
],
 "typeProperties": {
 "source": {
 "type": "<source type>"
 },
 "sink": {
 "type": "AzureBlobFSSink",
 "copyBehavior": "PreserveHierarchy"
 }
 }
 }
]
```

## Change data capture

Azure Data Factory can get new or changed files only from Azure Data Lake Storage Gen2 by enabling **Enable change data capture** in the mapping data flow source transformation. With this connector option, you can read new or updated files only and apply transformations before loading transformed data into destination datasets of your choice.

Make sure you keep the pipeline and activity name unchanged, so that the checkpoint can always be recorded from the last run to get changes from there. If you change your pipeline name or activity name, the checkpoint will be reset, and you will start from the beginning in the next run.

When you debug the pipeline, the **Enable change data capture** works as well. Be aware that the checkpoint will be reset when you refresh your browser during the debug run. After you are satisfied with the result from debug run, you can publish and trigger the pipeline. It will always start from the beginning regardless of the previous checkpoint recorded by debug run.

In the monitoring section, you always have the chance to rerun a pipeline. When you are doing so, the changes are always gotten from the checkpoint record in your selected pipeline run.

## Next steps

For a list of data stores supported as sources and sinks by the copy activity, see [Supported data stores](#).

# Use DistCp to copy data between Azure Storage Blobs and Azure Data Lake Storage Gen2

8/22/2022 • 4 minutes to read • [Edit Online](#)

You can use [DistCp](#) to copy data between a general purpose V2 storage account and a general purpose V2 storage account with hierarchical namespace enabled. This article provides instructions on how to use the DistCp tool.

DistCp provides a variety of command-line parameters and we strongly encourage you to read this article in order to optimize your usage of it. This article shows basic functionality while focusing on its use for copying data to a hierarchical namespace enabled account.

## Prerequisites

- An Azure subscription. For more information, see [Get Azure free trial](#).
- An existing Azure Storage account without Data Lake Storage Gen2 capabilities (hierarchical namespace) enabled.
- An Azure Storage account with Data Lake Storage Gen2 capabilities (hierarchical namespace) enabled. For instructions on how to create one, see [Create an Azure Storage account](#)
- A container that has been created in the storage account with hierarchical namespace enabled.
- An Azure HDInsight cluster with access to a storage account with the hierarchical namespace feature enabled. For more information, see [Use Azure Data Lake Storage Gen2 with Azure HDInsight clusters](#). Make sure you enable Remote Desktop for the cluster.

## Use DistCp from an HDInsight Linux cluster

An HDInsight cluster comes with the DistCp utility, which can be used to copy data from different sources into an HDInsight cluster. If you have configured the HDInsight cluster to use Azure Blob Storage and Azure Data Lake Storage together, the DistCp utility can be used out-of-the-box to copy data between as well. In this section, we look at how to use the DistCp utility.

1. Create an SSH session to your HDI cluster. For more information, see [Connect to a Linux-based HDInsight cluster](#).
2. Verify whether you can access your existing general purpose V2 account (without hierarchical namespace enabled).

```
hdfs dfs -ls wasbs://<container-name>@<storage-account-name>.blob.core.windows.net/
```

The output should provide a list of contents in the container.

3. Similarly, verify whether you can access the storage account with hierarchical namespace enabled from the cluster. Run the following command:

```
hdfs dfs -ls abfss://<container-name>@<storage-account-name>.dfs.core.windows.net/
```

The output should provide a list of files/folders in the Data Lake storage account.

4. Use DistCp to copy data from WASB to a Data Lake Storage account.

```
hadoop distcp wasbs://<container-name>@<storage-account-name>.blob.core.windows.net/example/data/gutenberg abfss://<container-name>@<storage-account-name>.dfs.core.windows.net/myfolder
```

The command copies the contents of the `/example/data/gutenberg/` folder in Blob storage to `/myfolder` in the Data Lake Storage account.

5. Similarly, use DistCp to copy data from Data Lake Storage account to Blob Storage (WASB).

```
hadoop distcp abfss://<container-name>@<storage-account-name>.dfs.core.windows.net/myfolder wasbs://<container-name>@<storage-account-name>.blob.core.windows.net/example/data/gutenberg
```

The command copies the contents of `/myfolder` in the Data Lake Store account to `/example/data/gutenberg/` folder in WASB.

## Performance considerations while using DistCp

Because DistCp's lowest granularity is a single file, setting the maximum number of simultaneous copies is the most important parameter to optimize it against Data Lake Storage. Number of simultaneous copies is equal to the number of mappers (`m`) parameter on the command line. This parameter specifies the maximum number of mappers that are used to copy data. Default value is 20.

### Example

```
hadoop distcp -m 100 wasbs://<container-name>@<storage-account-name>.blob.core.windows.net/example/data/gutenberg abfss://<container-name>@<storage-account-name>.dfs.core.windows.net/myfolder
```

### How do I determine the number of mappers to use?

Here's some guidance that you can use.

- **Step 1: Determine total memory available to the 'default' YARN app queue** - The first step is to determine the memory available to the 'default' YARN app queue. This information is available in the Ambari portal associated with the cluster. Navigate to YARN and view the Configs tab to see the YARN memory available to the 'default' app queue. This is the total available memory for your DistCp job (which is actually a MapReduce job).
- **Step 2: Calculate the number of mappers** - The value of `m` is equal to the quotient of total YARN memory divided by the YARN container size. The YARN container size information is available in the Ambari portal as well. Navigate to YARN and view the Configs tab. The YARN container size is displayed in this window. The equation to arrive at the number of mappers (`m`) is

$$m = (\text{number of nodes} * \text{YARN memory for each node}) / \text{YARN container size}$$

### Example

Let's assume that you have a 4x D14v2s cluster and you are trying to transfer 10 TB of data from 10 different folders. Each of the folders contains varying amounts of data and the file sizes within each folder are different.

- **Total YARN memory:** From the Ambari portal you determine that the YARN memory is 96 GB for a D14 node. So, total YARN memory for four node cluster is:  
  
YARN memory =  $4 * 96\text{GB} = 384\text{GB}$
- **Number of mappers:** From the Ambari portal you determine that the YARN container size is 3,072 MB for a D14 cluster node. So, number of mappers is:

$$m = (4 \text{ nodes} * 96\text{GB}) / 3072\text{MB} = 128 \text{ mappers}$$

If other applications are using memory, then you can choose to only use a portion of your cluster's YARN memory for DistCp.

### **Copying large datasets**

When the size of the dataset to be moved is large (for example, >1 TB) or if you have many different folders, you should consider using multiple DistCp jobs. There is likely no performance gain, but it spreads out the jobs so that if any job fails, you only need to restart that specific job rather than the entire job.

### **Limitations**

- DistCp tries to create mappers that are similar in size to optimize performance. Increasing the number of mappers may not always increase performance.
- DistCp is limited to only one mapper per file. Therefore, you should not have more mappers than you have files. Since DistCp can only assign one mapper to a file, this limits the amount of concurrency that can be used to copy large files.
- If you have a small number of large files, then you should split them into 256 MB file chunks to give you more potential concurrency.

# Transfer data with the Data Movement library

8/22/2022 • 11 minutes to read • [Edit Online](#)

The Azure Storage Data Movement library is a cross-platform open source library that is designed for high performance uploading, downloading, and copying of blobs and files. The Data Movement library provides convenient methods that aren't available in the Azure Storage client library for .NET. These methods provide the ability to set the number of parallel operations, track transfer progress, easily resume a canceled transfer, and much more.

This library also uses .NET Core, which means you can use it when building .NET apps for Windows, Linux and macOS. To learn more about .NET Core, refer to the [.NET Core documentation](#). This library also works for traditional .NET Framework apps for Windows.

This document demonstrates how to create a .NET Core console application that runs on Windows, Linux, and macOS and performs the following scenarios:

- Upload files and directories to Blob Storage.
- Define the number of parallel operations when transferring data.
- Track data transfer progress.
- Resume canceled data transfer.
- Copy file from URL to Blob Storage.
- Copy from Blob Storage to Blob Storage.

## Prerequisites

- [Visual Studio Code](#)
- An [Azure storage account](#)

## Setup

1. Visit the [.NET Core Installation Guide](#) to install the .NET Core SDK. When selecting your environment, choose the command-line option.
2. From the command line, create a directory for your project. Navigate into this directory, then type `dotnet new console -o <sample-project-name>` to create a C# console project.
3. Open this directory in Visual Studio Code. This step can be quickly done via the command line by typing `code .` in Windows.
4. Install the [C# extension](#) from the Visual Studio Code Marketplace. Restart Visual Studio Code.
5. At this point, you should see two prompts. One is for adding "required assets to build and debug." Click "yes." Another prompt is for restoring unresolved dependencies. Click "restore."
6. Modify `launch.json` under `.vscode` to use external terminal as a console. This setting should read as`"console": "externalTerminal"`
7. Visual Studio Code allows you to debug .NET Core applications. Hit `F5` to run your application and verify that your setup is working. You should see "Hello World!" printed to the console.

## Add the Data Movement library to your project

1. Add the latest version of the Data Movement library to the `dependencies` section of your `<project-name>.csproj` file. At the time of writing, this version would be

```
"Microsoft.Azure.Storage.DataMovement": "0.6.2"
```

2. A prompt should display to restore your project. Click the "restore" button. You can also restore your project from the command line by typing the command `dotnet restore` in the root of your project directory.

Modify `<project-name>.csproj`:

```
<Project Sdk="Microsoft.NET.Sdk">
 <PropertyGroup>
 <OutputType>Exe</OutputType>
 <TargetFramework>netcoreapp2.0</TargetFramework>
 </PropertyGroup>
 <ItemGroup>
 <PackageReference Include="Microsoft.Azure.Storage.DataMovement" Version="0.6.2" />
 </ItemGroup>
</Project>
```

## Set up the skeleton of your application

The first thing we do is set up the "skeleton" code of our application. This code prompts us for a Storage account name and account key and uses those credentials to create a `CloudStorageAccount` object. This object is used to interact with our Storage account in all transfer scenarios. The code also prompts us to choose the type of transfer operation we would like to execute.

Modify `Program.cs`:

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Diagnostics;
using Microsoft.Azure.Storage;
using Microsoft.Azure.Storage.Blob;
using Microsoft.Azure.Storage.DataMovement;

namespace DMLibSample
{
 public class Program
 {
 public static void Main()
 {
 Console.WriteLine("Enter Storage account name:");
 string accountName = Console.ReadLine();

 Console.WriteLine("\nEnter Storage account key:");
 string accountKey = Console.ReadLine();

 string storageConnectionString = "DefaultEndpointsProtocol=https;AccountName=" + accountName +
";AccountKey=" + accountKey;
 CloudStorageAccount account = CloudStorageAccount.Parse(storageConnectionString);

 ExecuteChoice(account);
 }

 public static void ExecuteChoice(CloudStorageAccount account)
 {
 Console.WriteLine("\nWhat type of transfer would you like to execute?\n1. Local file --> Azure
Blob\n2. Local directory --> Azure Blob directory\n3. URL (e.g. Amazon S3 file) --> Azure Blob\n4. Azure
Blob --> Azure Blob");
 int choice = int.Parse(Console.ReadLine());

 if(choice == 1)
 {
 TransferLocalFileToAzureBlob(account).Wait();
 }
 }
 }
}
```

```
 else if(choice == 2)
 {
 TransferLocalDirectoryToAzureBlobDirectory(account).Wait();
 }
 else if(choice == 3)
 {
 TransferUrlToAzureBlob(account).Wait();
 }
 else if(choice == 4)
 {
 TransferAzureBlobToAzureBlob(account).Wait();
 }
}

public static async Task TransferLocalFileToAzureBlob(CloudStorageAccount account)
{
}

public static async Task TransferLocalDirectoryToAzureBlobDirectory(CloudStorageAccount account)
{
}

public static async Task TransferUrlToAzureBlob(CloudStorageAccount account)
{
}

public static async Task TransferAzureBlobToAzureBlob(CloudStorageAccount account)
{
}
}
```

## Upload a local file to a blob

Add the methods `GetSourcePath` and `GetBlob` to `Program.cs`:

```
public static string GetSourcePath()
{
 Console.WriteLine("\nProvide path for source:");
 string sourcePath = Console.ReadLine();

 return sourcePath;
}

public static CloudBlockBlob GetBlob(CloudStorageAccount account)
{
 CloudBlobClient blobClient = account.CreateCloudBlobClient();

 Console.WriteLine("\nProvide name of Blob container:");
 string containerName = Console.ReadLine();
 CloudBlobContainer container = blobClient.GetContainerReference(containerName);
 container.CreateIfNotExistsAsync().Wait();

 Console.WriteLine("\nProvide name of new Blob:");
 string blobName = Console.ReadLine();
 CloudBlockBlob blob = container.GetBlockBlobReference(blobName);

 return blob;
}
```

Modify the `TransferLocalFileToAzureBlob` method:

```
public static async Task TransferLocalFileToAzureBlob(CloudStorageAccount account)
{
 string localFilePath = GetSourcePath();
 CloudBlockBlob blob = GetBlob(account);
 Console.WriteLine("\nTransfer started...");
 await TransferManager.UploadAsync(localFilePath, blob);
 Console.WriteLine("\nTransfer operation complete.");
 ExecuteChoice(account);
}
```

This code prompts us for the path to a local file, the name of a new or existing container, and the name of a new blob. The `TransferManager.UploadAsync` method performs the upload using this information.

Hit `F5` to run your application. You can verify that the upload occurred by viewing your Storage account with the [Microsoft Azure Storage Explorer](#).

## Set the number of parallel operations

One feature offered by the Data Movement library is the ability to set the number of parallel operations to increase the data transfer throughput. By default, the Data Movement library sets the number of parallel operations to  $8 * \text{the number of cores on your machine}$ .

Keep in mind that many parallel operations in a low-bandwidth environment may overwhelm the network connection and actually prevent operations from fully completing. You'll need to experiment with this setting to determine what works best based on your available network bandwidth.

Let's add some code that allows us to set the number of parallel operations. Let's also add code that times how long it takes for the transfer to complete.

Add a `SetNumberOfParallelOperations` method to `Program.cs`:

```
public static void SetNumberOfParallelOperations()
{
 Console.WriteLine("\nHow many parallel operations would you like to use?");
 string parallelOperations = Console.ReadLine();
 TransferManager.Configurations.ParallelOperations = int.Parse(parallelOperations);
}
```

Modify the `ExecuteChoice` method to use `SetNumberOfParallelOperations`:

```

public static void ExecuteChoice(CloudStorageAccount account)
{
 Console.WriteLine("\nWhat type of transfer would you like to execute?\n1. Local file --> Azure Blob\n2.
Local directory --> Azure Blob directory\n3. URL (e.g. Amazon S3 file) --> Azure Blob\n4. Azure Blob -->
Azure Blob");
 int choice = int.Parse(Console.ReadLine());

 SetNumberOfParallelOperations();

 if(choice == 1)
 {
 TransferLocalFileToAzureBlob(account).Wait();
 }
 else if(choice == 2)
 {
 TransferLocalDirectoryToAzureBlobDirectory(account).Wait();
 }
 else if(choice == 3)
 {
 TransferUrlToAzureBlob(account).Wait();
 }
 else if(choice == 4)
 {
 TransferAzureBlobToAzureBlob(account).Wait();
 }
}

```

Modify the `TransferLocalFileToAzureBlob` method to use a timer:

```

public static async Task TransferLocalFileToAzureBlob(CloudStorageAccount account)
{
 string localFilePath = GetSourcePath();
 CloudBlockBlob blob = GetBlob(account);
 Console.WriteLine("\nTransfer started...");
 Stopwatch stopWatch = Stopwatch.StartNew();
 await TransferManager.UploadAsync(localFilePath, blob);
 stopWatch.Stop();
 Console.WriteLine("\nTransfer operation completed in " + stopWatch.Elapsed.TotalSeconds + " seconds.");
 ExecuteChoice(account);
}

```

## Track transfer progress

Knowing how long it took for the data to transfer is helpful. However, being able to see the progress of the transfer *during* the transfer operation would be even better. To achieve this scenario, we need to create a `TransferContext` object. The `TransferContext` object comes in two forms: `SingleTransferContext` and `DirectoryTransferContext`. The former is for transferring a single file and the latter is for transferring a directory of files.

Add the methods `GetSingleTransferContext` and `GetDirectoryTransferContext` to `Program.cs`:

```

public static SingleTransferContext GetSingleTransferContext(TransferCheckpoint checkpoint)
{
 SingleTransferContext context = new SingleTransferContext(checkpoint);

 context.ProgressHandler = new Progress<TransferStatus>((progress) =>
 {
 Console.WriteLine("\rBytes transferred: {0}", progress.BytesTransferred);
 });

 return context;
}

public static DirectoryTransferContext GetDirectoryTransferContext(TransferCheckpoint checkpoint)
{
 DirectoryTransferContext context = new DirectoryTransferContext(checkpoint);

 context.ProgressHandler = new Progress<TransferStatus>((progress) =>
 {
 Console.WriteLine("\rBytes transferred: {0}", progress.BytesTransferred);
 });

 return context;
}

```

Modify the `TransferLocalFileToAzureBlob` method to use `GetSingleTransferContext`:

```

public static async Task TransferLocalFileToAzureBlob(CloudStorageAccount account)
{
 string localFilePath = GetSourcePath();
 CloudBlockBlob blob = GetBlob(account);
 TransferCheckpoint checkpoint = null;
 SingleTransferContext context = GetSingleTransferContext(checkpoint);
 Console.WriteLine("\nTransfer started...\n");
 Stopwatch stopWatch = Stopwatch.StartNew();
 await TransferManager.UploadAsync(localFilePath, blob, null, context);
 stopWatch.Stop();
 Console.WriteLine("\nTransfer operation completed in " + stopWatch.Elapsed.TotalSeconds + " seconds.");
 ExecuteChoice(account);
}

```

## Resume a canceled transfer

Another convenient feature offered by the Data Movement library is the ability to resume a canceled transfer. Let's add some code that allows us to temporarily cancel the transfer by typing `c`, and then resume the transfer 3 seconds later.

Modify `TransferLocalFileToAzureBlob`:

```

public static async Task TransferLocalFileToAzureBlob(CloudStorageAccount account)
{
 string localFilePath = GetSourcePath();
 CloudBlockBlob blob = GetBlob(account);
 TransferCheckpoint checkpoint = null;
 SingleTransferContext context = GetSingleTransferContext(checkpoint);
 CancellationTokenSource cancellationSource = new CancellationTokenSource();
 Console.WriteLine("\nTransfer started...\nPress 'c' to temporarily cancel your transfer...\n");

 Stopwatch stopWatch = Stopwatch.StartNew();
 Task task;
 ConsoleKeyInfo keyinfo;
 try
 {
 task = TransferManager.UploadAsync(localFilePath, blob, null, context, cancellationSource.Token);
 while(!task.IsCompleted)
 {
 if(Console.KeyAvailable)
 {
 keyinfo = Console.ReadKey(true);
 if(keyinfo.Key == ConsoleKey.C)
 {
 cancellationSource.Cancel();
 }
 }
 await task;
 }
 }
 catch(Exception e)
 {
 Console.WriteLine("\nThe transfer is canceled: {0}", e.Message);
 }

 if(cancellationSource.IsCancellationRequested)
 {
 Console.WriteLine("\nTransfer will resume in 3 seconds...");
 Thread.Sleep(3000);
 checkpoint = context.LastCheckpoint;
 context = GetSingleTransferContext(checkpoint);
 Console.WriteLine("\nResuming transfer...\n");
 await TransferManager.UploadAsync(localFilePath, blob, null, context);
 }

 stopWatch.Stop();
 Console.WriteLine("\nTransfer operation completed in " + stopWatch.Elapsed.TotalSeconds + " seconds.");
 ExecuteChoice(account);
}

```

Up until now, our `checkpoint` value has always been set to `null`. Now, if we cancel the transfer, we retrieve the last checkpoint of our transfer, then use this new checkpoint in our transfer context.

## Transfer a local directory to Blob storage

It would be disappointing if the Data Movement library could only transfer one file at a time. Luckily, this is not the case. The Data Movement library provides the ability to transfer a directory of files and all of its subdirectories. Let's add some code that allows us to do just that.

First, add the method `GetBlobDirectory` to `Program.cs`:

```
public static CloudBlobDirectory GetBlobDirectory(CloudStorageAccount account)
{
 CloudBlobClient blobClient = account.CreateCloudBlobClient();

 Console.WriteLine("\nProvide name of Blob container. This can be a new or existing Blob container:");
 string containerName = Console.ReadLine();
 CloudBlobContainer container = blobClient.GetContainerReference(containerName);
 container.CreateIfNotExistsAsync().Wait();

 CloudBlobDirectory blobDirectory = container.GetDirectoryReference("");
 return blobDirectory;
}
```

Then, modify `TransferLocalDirectoryToAzureBlobDirectory` :

```

public static async Task TransferLocalDirectoryToAzureBlobDirectory(CloudStorageAccount account)
{
 string localDirectoryPath = GetSourcePath();
 CloudBlobDirectory blobDirectory = GetBlobDirectory(account);
 TransferCheckpoint checkpoint = null;
 DirectoryTransferContext context = GetDirectoryTransferContext(checkpoint);
 CancellationTokenSource cancellationSource = new CancellationTokenSource();
 Console.WriteLine("\nTransfer started...\nPress 'c' to temporarily cancel your transfer...\n");

 Stopwatch stopWatch = Stopwatch.StartNew();
 Task task;
 ConsoleKeyInfo keyinfo;
 UploadDirectoryOptions options = new UploadDirectoryOptions()
 {
 Recursive = true
 };

 try
 {
 task = TransferManager.UploadDirectoryAsync(localDirectoryPath, blobDirectory, options, context,
cancellationSource.Token);
 while(!task.IsCompleted)
 {
 if(Console.KeyAvailable)
 {
 keyinfo = Console.ReadKey(true);
 if(keyinfo.Key == ConsoleKey.C)
 {
 cancellationSource.Cancel();
 }
 }
 }
 await task;
 }
 catch(Exception e)
 {
 Console.WriteLine("\nThe transfer is canceled: {0}", e.Message);
 }

 if(cancellationSource.IsCancellationRequested)
 {
 Console.WriteLine("\nTransfer will resume in 3 seconds...");
 Thread.Sleep(3000);
 checkpoint = context.LastCheckpoint;
 context = GetDirectoryTransferContext(checkpoint);
 Console.WriteLine("\nResuming transfer...\n");
 await TransferManager.UploadDirectoryAsync(localDirectoryPath, blobDirectory, options, context);
 }

 stopWatch.Stop();
 Console.WriteLine("\nTransfer operation completed in " + stopWatch.Elapsed.TotalSeconds + " seconds.");
 ExecuteChoice(account);
}

```

There are a few differences between this method and the method for uploading a single file. We're now using `TransferManager.UploadDirectoryAsync` and the `GetDirectoryTransferContext` method we created earlier. In addition, we now provide an `options` value to our upload operation, which allows us to indicate that we want to include subdirectories in our upload.

## Copy a file from URL to a blob

Now, let's add code that allows us to copy a file from a URL to an Azure Blob.

Modify `TransferUrlToAzureBlob`:

```

public static async Task TransferUrlToAzureBlob(CloudStorageAccount account)
{
 Uri uri = new Uri(GetSourcePath());
 CloudBlockBlob blob = GetBlob(account);
 TransferCheckpoint checkpoint = null;
 SingleTransferContext context = GetSingleTransferContext(checkpoint);
 CancellationTokenSource cancellationSource = new CancellationTokenSource();
 Console.WriteLine("\nTransfer started...\nPress 'c' to temporarily cancel your transfer...\n");

 Stopwatch stopWatch = Stopwatch.StartNew();
 Task task;
 ConsoleKeyInfo keyinfo;
 try
 {
 task = TransferManager.CopyAsync(uri, blob, true, null, context, cancellationSource.Token);
 while(!task.IsCompleted)
 {
 if(Console.KeyAvailable)
 {
 keyinfo = Console.ReadKey(true);
 if(keyinfo.Key == ConsoleKey.C)
 {
 cancellationSource.Cancel();
 }
 }
 await task;
 }
 }
 catch(Exception e)
 {
 Console.WriteLine("\nThe transfer is canceled: {0}", e.Message);
 }

 if(cancellationSource.IsCancellationRequested)
 {
 Console.WriteLine("\nTransfer will resume in 3 seconds...");
 Thread.Sleep(3000);
 checkpoint = context.LastCheckpoint;
 context = GetSingleTransferContext(checkpoint);
 Console.WriteLine("\nResuming transfer...\n");
 await TransferManager.CopyAsync(uri, blob, true, null, context, cancellationSource.Token);
 }

 stopWatch.Stop();
 Console.WriteLine("\nTransfer operation completed in " + stopWatch.Elapsed.TotalSeconds + " seconds.");
 ExecuteChoice(account);
}

```

One important use case for this feature is when you need to move data from another cloud service (e.g. AWS) to Azure. As long as you have a URL that gives you access to the resource, you can easily move that resource into Azure Blobs by using the `TransferManager.CopyAsync` method. This method also introduces a new boolean parameter. Setting this parameter to `true` indicates that we want to do an asynchronous server-side copy. Setting this parameter to `false` indicates a synchronous copy - meaning the resource is downloaded to our local machine first, then uploaded to Azure Blob. However, synchronous copy is currently only available for copying from one Azure Storage resource to another.

## Copy a blob

Another feature that's uniquely provided by the Data Movement library is the ability to copy from one Azure Storage resource to another.

Modify `TransferAzureBlobToAzureBlob`:

```

public static async Task TransferAzureBlobToAzureBlob(CloudStorageAccount account)
{
 CloudBlockBlob sourceBlob = GetBlob(account);
 CloudBlockBlob destinationBlob = GetBlob(account);
 TransferCheckpoint checkpoint = null;
 SingleTransferContext context = GetSingleTransferContext(checkpoint);
 CancellationTokenSource cancellationSource = new CancellationTokenSource();
 Console.WriteLine("\nTransfer started...\nPress 'c' to temporarily cancel your transfer...\n");

 Stopwatch stopWatch = Stopwatch.StartNew();
 Task task;
 ConsoleKeyInfo keyinfo;
 try
 {
 task = TransferManager.CopyAsync(sourceBlob, destinationBlob, CopyMethod.ServiceSideAsyncCopy, null,
context, cancellationSource.Token);
 while(!task.IsCompleted)
 {
 if(Console.KeyAvailable)
 {
 keyinfo = Console.ReadKey(true);
 if(keyinfo.Key == ConsoleKey.C)
 {
 cancellationSource.Cancel();
 }
 }
 }
 await task;
 }
 catch(Exception e)
 {
 Console.WriteLine("\nThe transfer is canceled: {0}", e.Message);
 }

 if(cancellationSource.IsCancellationRequested)
 {
 Console.WriteLine("\nTransfer will resume in 3 seconds...");
 Thread.Sleep(3000);
 checkpoint = context.LastCheckpoint;
 context = GetSingleTransferContext(checkpoint);
 Console.WriteLine("\nResuming transfer...\n");
 await TransferManager.CopyAsync(sourceBlob, destinationBlob, false, null, context,
cancellationSource.Token);
 }

 stopWatch.Stop();
 Console.WriteLine("\nTransfer operation completed in " + stopWatch.Elapsed.TotalSeconds + " seconds.");
 ExecuteChoice(account);
}

```

In this example, we set the boolean parameter in `TransferManager.CopyAsync` to `false` to indicate that we want to do a synchronous copy. This means that the resource is downloaded to our local machine first, then uploaded to Azure Blob. The synchronous copy option is a great way to ensure that your copy operation has a consistent speed. In contrast, the speed of an asynchronous server-side copy is dependent on the available network bandwidth on the server, which can fluctuate. However, synchronous copy may generate additional egress cost compared to asynchronous copy. The recommended approach is to use synchronous copy in an Azure VM that is in the same region as your source storage account to avoid egress cost.

The data movement application is now complete. [The full code sample is available on GitHub.](#)

## Next steps

[Azure Storage Data Movement library reference documentation.](#)

**TIP**

Manage Azure Blob storage resources with Azure Storage Explorer. [Azure Storage Explorer](#) is a free, standalone app from Microsoft that enables you to [manage Azure Blob storage resources](#). Using Azure Storage Explorer, you can visually create, read, update, and delete blob containers and blobs, as well as manage access to your blobs containers and blobs.

# Use the Azure portal to manage ACLs in Azure Data Lake Storage Gen2

8/22/2022 • 2 minutes to read • [Edit Online](#)

This article shows you how to use [Azure portal](#) to manage the access control list (ACL) of a directory or blob in storage accounts that have the hierarchical namespace feature enabled on them.

For information about the structure of the ACL, see [Access control lists \(ACLs\) in Azure Data Lake Storage Gen2](#).

To learn about how to use ACLs and Azure roles together, see [Access control model in Azure Data Lake Storage Gen2](#).

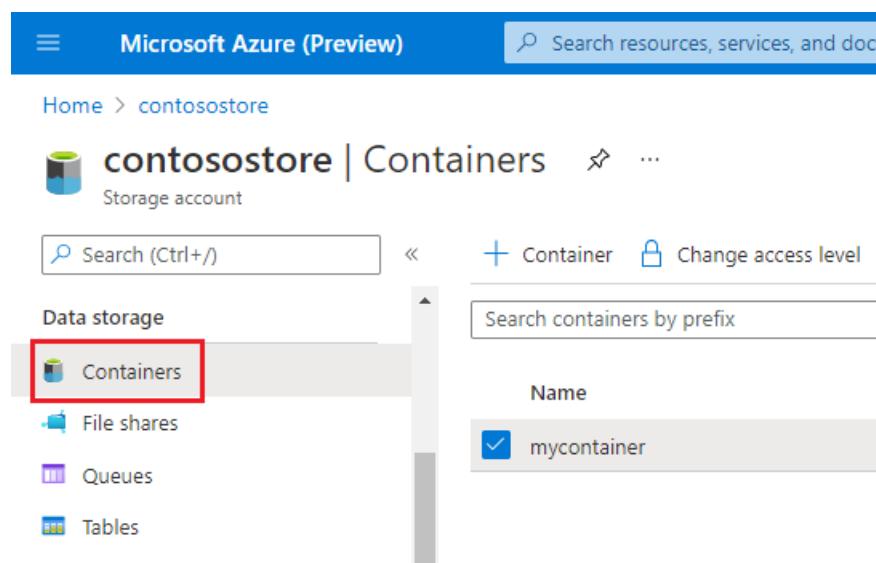
## Prerequisites

- An Azure subscription. See [Get Azure free trial](#).
- A storage account that has the hierarchical namespace feature enabled on it. Follow [these](#) instructions to create one.
- You must have one of the following security permissions:
  - Your user identity has been assigned the [Storage Blob Data Owner](#) role in the scope of either the target container, storage account, parent resource group or subscription.
  - You are the owning user of the target container, directory, or blob to which you plan to apply ACL settings.

## Manage an ACL

1. Sign in to the [Azure portal](#) to get started.
2. Locate your storage account and display the account overview.
3. Select **Containers** under **Data storage**.

The containers in the storage account appear.



4. Navigate to any container, directory, or blob. Right-click the object, and then select **Manage ACL**.

Microsoft Azure

Search resources, services, and docs (G+/)

Home > contosostore >

## mycontainer

Container

Search (Ctrl+ /) <

Upload Add Directory Refresh Rename

- Overview
- Diagnose and solve problems
- Access Control (IAM)

### Settings

- Shared access signature
- Manage ACL
- Access policy
- Properties
- Metadata
- Editor (preview)

Authentication method: Access key (Switch to Azure AD User Account)

Location: mycontainer

Search blobs by prefix (case-sensitive)

Name	Modified
myDirectory	

- Rename
- Properties
- Generate SAS
- Manage ACL
- Delete

The **Access permissions** tab of the **Manage ACL** page appears. Use the controls in this tab to manage access to the object.

Microsoft Azure

Search resources, services, and docs (G+/)

Home > contosostore > mycontainer >

## Manage ACL

X

container: mycontainer (storage account: contosostore)

Set and manage permissions for:  
/myDirectory

[Learn more about access control lists \(ACLs\)](#)

[Access permissions](#)    [Default permissions](#)

[+ Add principal](#)    [+ Add mask](#)

Security principal	Read	Write	Execute	
Owner: \$superuser	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<a href="#">Edit</a>
Owning group: \$superuser	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<a href="#">Edit</a>
Other	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

i Read and write permissions will only work for a security principal if the security principal also has execute permissions on all parent directories, including the container (root directory).

[Save](#)

[Discard](#)

5. To add a *security principal* to the ACL, select the **Add principal** button.

#### TIP

A security principal is an object that represents a user, group, service principal, or managed identity that is defined in Azure Active Directory (AD).

Find the security principal by using the search box, and then click the **Select** button.

The screenshot shows the 'Add principal' dialog box in the Microsoft Azure portal. The search bar contains 'contoso'. Below the search bar, a list displays a single result: 'Contoso' with the email 'contoso@contoso.com' and the status 'Selected'. A red box highlights both the search bar and the list item. At the bottom right of the dialog, there is a red-bordered 'Select' button. The left sidebar shows the navigation path: Home > contosostore > mycontainer > Manage ACL. The 'Default permissions' tab is selected. A note at the bottom states: 'Read and write permissions will only work for a single directory, including the container (root directory)'.

#### NOTE

We recommend that you create a security group in Azure AD, and then maintain permissions on the group rather than for individual users. For details on this recommendation, as well as other best practices, see [Access control model in Azure Data Lake Storage Gen2](#).

6. To manage the *default ACL*, select the **default permissions** tab, and then select the **Configure default permissions** checkbox.

#### TIP

A default ACL is a template of an ACL that determines the access ACLs for any child items that are created under a directory. A blob doesn't have a default ACL, so this tab appears only for directories.

The screenshot shows the Microsoft Azure portal with the URL [https://portal.azure.com/#blade/HubsBlade](#). The page title is "Manage ACL" under "mycontainer > /myDirectory". The top navigation bar includes "Search resources, services, and docs (G+)" and three dots for more options. The main content area has a heading "Set and manage permissions for: /myDirectory" and a link "Learn more about access control lists (ACLs)". Below this, there are two tabs: "Access permissions" and "Default permissions", with "Default permissions" selected and highlighted with a red box. A note states: "The default ACL determines permissions for new children of this directory. Changing the default ACL does not affect children that already exist." A link "Learn more about default ACLs" is provided. A checkbox "Configure default permissions" is checked and highlighted with a red box. Below this are buttons "+ Add principal" and "+ Add mask". A table follows, showing permissions for "Owner", "Owning group", and "Other" security principals across "Read", "Write", and "Execute" columns. All checkboxes are checked except for "Other" under "Write". A tooltip indicates: "Read and write permissions will only work for a security principal if the security principal also has execute permissions on all parent directories, including the container (root directory)." At the bottom are "Save" and "Discard" buttons.

## Apply an ACL recursively

You can apply ACL entries recursively on the existing child items of a parent directory without having to make these changes individually for each child item. However, you can't apply ACL entries recursively by using the Azure portal.

To apply ACLs recursively, use Azure Storage Explorer, PowerShell, or the Azure CLI. If you prefer to write code, you can also use the .NET, Java, Python, or Node.js APIs.

You can find the complete list of guides here: [How to set ACLs](#).

## Next steps

Learn about the Data Lake Storage Gen2 permission model.

[Access control model in Azure Data Lake Storage Gen2](#)

# Use Azure Storage Explorer to manage ACLs in Azure Data Lake Storage Gen2

8/22/2022 • 3 minutes to read • [Edit Online](#)

This article shows you how to use [Azure Storage Explorer](#) to manage access control lists (ACLs) in storage accounts that has hierarchical namespace (HNS) enabled.

You can use Storage Explorer to view, and then update the ACLs of directories and files. ACL inheritance is already available for new child items that are created under a parent directory. But you can also apply ACL settings recursively on the existing child items of a parent directory without having to make these changes individually for each child item.

This article shows you how to modify the ACL of file or directory and how to apply ACL settings recursively to child directories.

## Prerequisites

- An Azure subscription. See [Get Azure free trial](#).
- A storage account that has hierarchical namespace (HNS) enabled. Follow [these](#) instructions to create one.
- Azure Storage Explorer installed on your local computer. To install Azure Storage Explorer for Windows, Macintosh, or Linux, see [Azure Storage Explorer](#).
- You must have one of the following security permissions:
  - Your user identity has been assigned the [Storage Blob Data Owner](#) role in the scope of the either the target container, storage account, parent resource group or subscription.
  - You are the owning user of the target container, directory, or blob to which you plan to apply ACL settings.

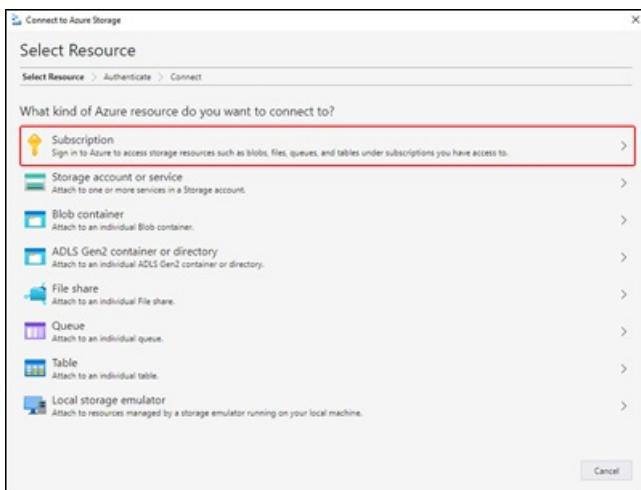
### NOTE

Storage Explorer makes use of both the Blob (blob) & Data Lake Storage Gen2 (dfs) [endpoints](#) when working with Azure Data Lake Storage Gen2. If access to Azure Data Lake Storage Gen2 is configured using private endpoints, ensure that two private endpoints are created for the storage account: one with the target sub-resource `blob` and the other with the target sub-resource `dfs`.

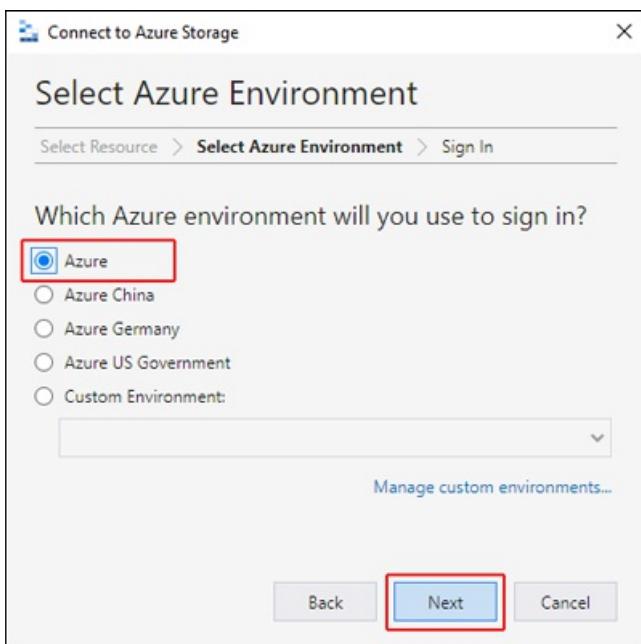
## Sign in to Storage Explorer

When you first start Storage Explorer, the [Microsoft Azure Storage Explorer - Connect to Azure Storage](#) window appears. While Storage Explorer provides several ways to connect to storage accounts, only one way is currently supported for managing ACLs.

In the **Select Resource** panel, select **Subscription**.

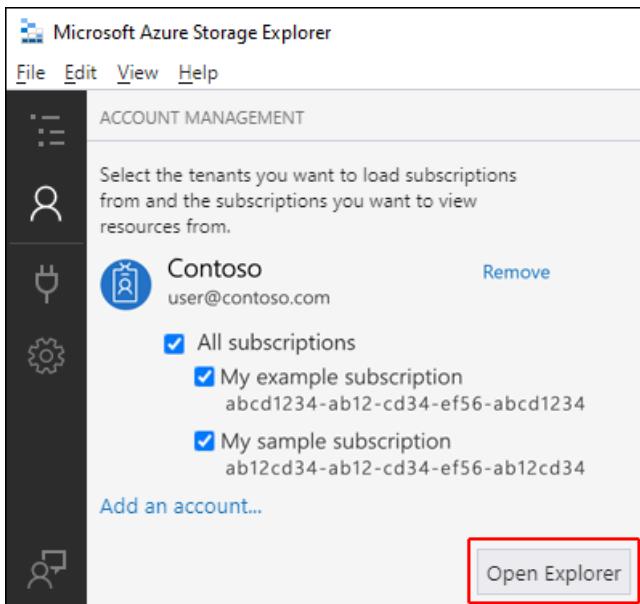


In the **Select Azure Environment** panel, select an Azure environment to sign in to. You can sign in to global Azure, a national cloud or an Azure Stack instance. Then select **Next**.

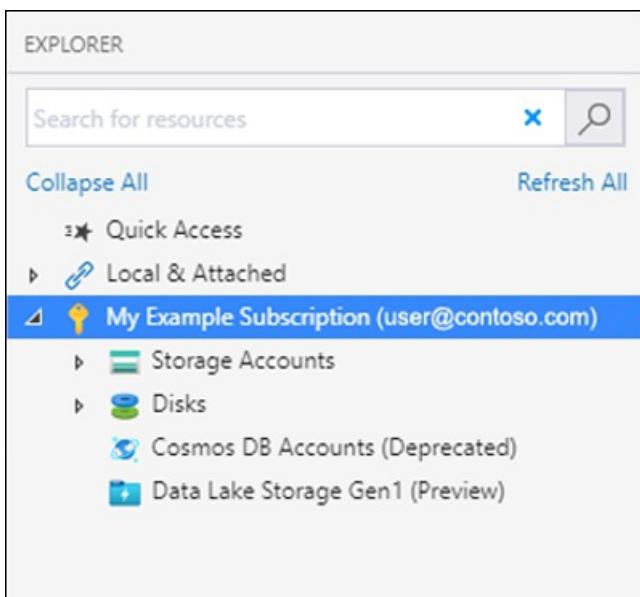


Storage Explorer will open a webpage for you to sign in.

After you successfully sign in with an Azure account, the account and the Azure subscriptions associated with that account appear under **ACCOUNT MANAGEMENT**. Select the Azure subscriptions that you want to work with, and then select **Open Explorer**.



When it completes connecting, Azure Storage Explorer loads with the **Explorer** tab shown. This view gives you insight to all of your Azure storage accounts as well as local storage configured through the [Azurite storage emulator](#) or [Azure Stack](#) environments.



## Manage an ACL

Right-click the container, a directory, or a file, and then click **Manage Access Control Lists**. The following screenshot shows the menu as it appears when you right-click a directory.

The screenshot shows the Azure Storage Explorer interface. At the top, there's a toolbar with icons for Upload, Download, Open, New Folder, Select All, Copy, Paste, Rename, Move, and Manage ACLs. Below the toolbar, the path 'my-container' is displayed. In the main pane, there's a table with columns: Name, Access Tier, Access Tier Last Modified, and Last Modified. Two items are listed: 'my-directory' and 'my-parent-directory'. A context menu is open over 'my-parent-directory', listing options: Open, Download, Copy, Paste, Change Access Tier..., Manage Access Control Lists..., Propagate Access Control Lists..., and Acquire Lease. The 'Manage Access Control Lists...' option is highlighted with a red box.

The **Manage Access** dialog box allows you to manage permissions for owner and the owners group. It also allows you to add new users and groups to the access control list for whom you can then manage permissions.

The screenshot shows the 'Manage Access' dialog box. At the top, it says 'Manage Access' and 'Managing permissions for: my-container/my-parent-directory'. It includes a link to 'Learn more about access control lists (ACLs)'. Below this, a section titled 'Users, groups, and service principals:' lists '\$superuser (Owner)' and '\$superuser (Owning Group)'. There are 'Edit' icons next to each entry. Underneath, there are buttons for 'Other' and 'Mask'. A 'Add' button is located below the list. The main area shows 'Permissions for: \$superuser' with three columns: Read, Write, and Execute. Each column has a checkbox. The 'Access' row has all checkboxes checked. The 'Default \*' row also has all checkboxes checked. A note below states: 'Read and write permissions will only work for an entity if the entity also has execute permissions on all parent directories, including the container (root directory).'. At the bottom, there's a note: '\* The default ACL determines permissions for new children of this directory. Changing the default ACL does not affect children that already exist. Learn more about default ACLs.' Finally, there are 'OK' and 'Cancel' buttons at the bottom right.

To add a new user or group to the access control list, select the **Add** button. Then, enter the corresponding Azure Active Directory (Azure AD) entry you wish to add to the list and then select **Add**. The user or group will now appear in the **Users and groups:** field, allowing you to begin managing their permissions.

#### NOTE

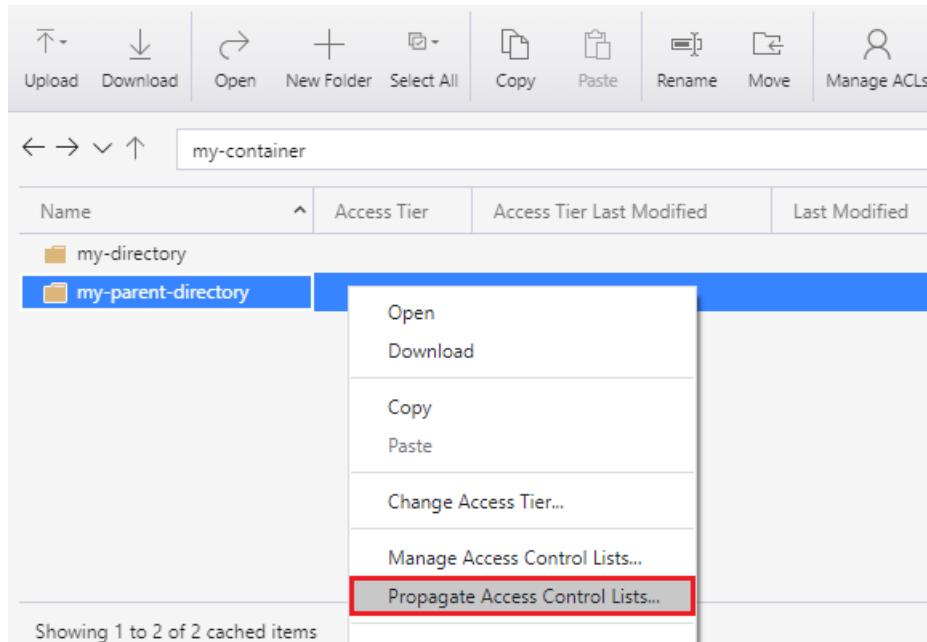
It is a best practice, and recommended, to create a security group in Azure AD and maintain permissions on the group rather than individual users. For details on this recommendation, as well as other best practices, see [Access control model in Azure Data Lake Storage Gen2](#).

Use the check box controls to set access and default ACLs. To learn more about the difference between these types of ACLs, see [Types of ACLs](#).

## Apply ACLs recursively

You can apply ACL entries recursively on the existing child items of a parent directory without having to make these changes individually for each child item.

To apply ACL entries recursively, Right-click the container or a directory, and then click **Propagate Access Control Lists**. The following screenshot shows the menu as it appears when you right-click a directory.



## Next steps

Learn about the Data Lake Storage Gen2 permission model.

[Access control model in Azure Data Lake Storage Gen2](#)

# Use PowerShell to manage ACLs in Azure Data Lake Storage Gen2

8/22/2022 • 11 minutes to read • [Edit Online](#)

This article shows you how to use PowerShell to get, set, and update the access control lists of directories and files.

ACL inheritance is already available for new child items that are created under a parent directory. But you can also add, update, and remove ACLs recursively on the existing child items of a parent directory without having to make these changes individually for each child item.

[Reference](#) | [Give feedback](#)

## Prerequisites

- An Azure subscription. For more information, see [Get Azure free trial](#).
- A storage account that has hierarchical namespace (HNS) enabled. Follow [these](#) instructions to create one.
- Azure CLI version [2.6.0](#) or higher.
- One of the following security permissions:
  - A provisioned Azure Active Directory (AD) [security principal](#) that has been assigned the [Storage Blob Data Owner](#) role in the scope of either the target container, parent resource group or subscription.
  - Owning user of the target container or directory to which you plan to apply ACL settings. To set ACLs recursively, this includes all child items in the target container or directory.
  - Storage account key.

## Install the PowerShell module

1. Verify that the version of PowerShell that have installed is [5.1](#) or higher by using the following command.

```
echo $PSVersionTable.PSVersion.ToString()
```

To upgrade your version of PowerShell, see [Upgrading existing Windows PowerShell](#)

2. Install **Az.Storage** module.

```
Install-Module Az.Storage -Repository PSGallery -Force
```

For more information about how to install PowerShell modules, see [Install the Azure PowerShell module](#)

## Connect to the account

Choose how you want your commands to obtain authorization to the storage account.

## Option 1: Obtain authorization by using Azure Active Directory (AD)

### NOTE

If you're using Azure Active Directory (Azure AD) to authorize access, then make sure that your security principal has been assigned the [Storage Blob Data Owner role](#). To learn more about how ACL permissions are applied and the effects of changing them, see [Access control model in Azure Data Lake Storage Gen2](#).

With this approach, the system ensures that your user account has the appropriate Azure role-based access control (Azure RBAC) assignments and ACL permissions.

1. Open a Windows PowerShell command window, and then sign in to your Azure subscription with the `Connect-AzAccount` command and follow the on-screen directions.

```
Connect-AzAccount
```

2. If your identity is associated with more than one subscription, then set your active subscription to subscription of the storage account that you want create and manage directories in. In this example, replace the `<subscription-id>` placeholder value with the ID of your subscription.

```
Select-AzSubscription -SubscriptionId <subscription-id>
```

3. Get the storage account context.

```
$ctx = New-AzStorageContext -StorageAccountName '<storage-account-name>' -UseConnectedAccount
```

## Option 2: Obtain authorization by using the storage account key

With this approach, the system doesn't check Azure RBAC or ACL permissions. Get the storage account context by using an account key.

```
$ctx = New-AzStorageContext -StorageAccountName '<storage-account-name>' -StorageAccountKey '<storage-account-key>'
```

## Get ACLs

Get the ACL of a directory or file by using the `Get-AzDataLakeGen2Item` cmdlet.

This example gets the ACL of the root directory of a **container** and then prints the ACL to the console.

```
$filesystemName = "my-file-system"
$filesystem = Get-AzDataLakeGen2Item -Context $ctx -FileSystem $filesystemName
$filesystem.ACL
```

This example gets the ACL of a **directory**, and then prints the ACL to the console.

```
$filesystemName = "my-file-system"
$dirname = "my-directory/"
$dir = Get-AzDataLakeGen2Item -Context $ctx -FileSystem $filesystemName -Path $dirname
$dir.ACL
```

This example gets the ACL of a **file** and then prints the ACL to the console.

```
$filePath = "my-directory/upload.txt"
$file = Get-AzDataLakeGen2Item -Context $ctx -FileSystem $filesystemName -Path $filePath
$file.ACL
```

The following image shows the output after getting the ACL of a directory.

DefaultScope	AccessControlType	EntityId	Permissions
False	User		rwx
False	Group		r-x
False	Other		---

In this example, the owning user has read, write, and execute permissions. The owning group has only read and execute permissions. For more information about access control lists, see [Access control in Azure Data Lake Storage Gen2](#).

## Set ACLs

When you *set* an ACL, you *replace* the entire ACL including all of its entries. If you want to change the permission level of a security principal or add a new security principal to the ACL without affecting other existing entries, you should *update* the ACL instead. To update an ACL instead of replace it, see the [Update ACLs](#) section of this article.

If you choose to *set* the ACL, you must add an entry for the owning user, an entry for the owning group, and an entry for all other users. To learn more about the owning user, the owning group, and all other users, see [Users and identities](#).

This section shows you how to:

- Set an ACL
- Set ACLs recursively

### Set an ACL

Use the `set-AzDataLakeGen2ItemAclObject` cmdlet to create an ACL for the owning user, owning group, or other users. Then, use the `Update-AzDataLakeGen2Item` cmdlet to commit the ACL.

This example sets the ACL on the root directory of a **container** for the owning user, owning group, or other users, and then prints the ACL to the console.

```
$filesystemName = "my-file-system"
$acl = set-AzDataLakeGen2ItemAclObject -AccessControlType user -Permission rw-
$acl = set-AzDataLakeGen2ItemAclObject -AccessControlType group -Permission rw- -InputObject $acl
$acl = set-AzDataLakeGen2ItemAclObject -AccessControlType other -Permission -wx -InputObject $acl
Update-AzDataLakeGen2Item -Context $ctx -FileSystem $filesystemName -Acl $acl
$filesystem = Get-AzDataLakeGen2Item -Context $ctx -FileSystem $filesystemName
$filesystem.ACL
```

This example sets the ACL on a **directory** for the owning user, owning group, or other users, and then prints the ACL to the console.

```

$filesystemName = "my-file-system"
$dirname = "my-directory/"
$acl = set-AzDataLakeGen2ItemAclObject -AccessControlType user -Permission rw-
$acl = set-AzDataLakeGen2ItemAclObject -AccessControlType group -Permission rw- -InputObject $acl
$acl = set-AzDataLakeGen2ItemAclObject -AccessControlType other -Permission -wx -InputObject $acl
Update-AzDataLakeGen2Item -Context $ctx -FileSystem $filesystemName -Path $dirname -Acl $acl
$dir = Get-AzDataLakeGen2Item -Context $ctx -FileSystem $filesystemName -Path $dirname
$dir.ACL

```

#### NOTE

If you want to set a default ACL entry, use the **-DefaultScope** parameter when you run the **Set-AzDataLakeGen2ItemAclObject** command. For example:

```
$acl = set-AzDataLakeGen2ItemAclObject -AccessControlType user -Permission rwx -DefaultScope .
```

This example sets the ACL on a file for the owning user, owning group, or other users, and then prints the ACL to the console.

```

$filesystemName = "my-file-system"
$filePath = "my-directory/upload.txt"
$acl = set-AzDataLakeGen2ItemAclObject -AccessControlType user -Permission rw-
$acl = set-AzDataLakeGen2ItemAclObject -AccessControlType group -Permission rw- -InputObject $acl
$acl = set-AzDataLakeGen2ItemAclObject -AccessControlType other -Permission "-wx" -InputObject $acl
Update-AzDataLakeGen2Item -Context $ctx -FileSystem $filesystemName -Path $filePath -Acl $acl
$file = Get-AzDataLakeGen2Item -Context $ctx -FileSystem $filesystemName -Path $filePath
$file.ACL

```

#### NOTE

To set the ACL of a specific group or user, use their respective object IDs. For example,

```
group:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx or user:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx .
```

The following image shows the output after setting the ACL of a file.

DefaultScope	AccessControlType	EntityId	Permissions
False	User		rw-
False	Group		rw-
False	Other		-wx

In this example, the owning user and owning group have only read and write permissions. All other users have write and execute permissions. For more information about access control lists, see [Access control in Azure Data Lake Storage Gen2](#).

#### Set ACLs recursively

Set ACLs recursively by using the **Set-AzDataLakeGen2AclRecursive** cmdlet.

This example sets the ACL of a directory named `my-parent-directory`. These entries give the owning user read, write, and execute permissions, gives the owning group only read and execute permissions, and gives all others no access. The last ACL entry in this example gives a specific user with the object ID "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx" read and execute permissions.

```

$filesystemName = "my-container"
$dirname = "my-parent-directory/"
$userID = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";

$acl = Set-AzDataLakeGen2ItemAclObject -AccessControlType user -Permission rwx
$acl = Set-AzDataLakeGen2ItemAclObject -AccessControlType group -Permission r-x -InputObject $acl
$acl = Set-AzDataLakeGen2ItemAclObject -AccessControlType other -Permission "---" -InputObject $acl
$acl = Set-AzDataLakeGen2ItemAclObject -AccessControlType user -EntityId $userID -Permission r-x -
InputObject $acl

Set-AzDataLakeGen2AclRecursive -Context $ctx -FileSystem $filesystemName -Path $dirname -Acl $acl

```

#### NOTE

If you want to set a **default** ACL entry, use the **-DefaultScope** parameter when you run the **Set-AzDataLakeGen2ItemAclObject** command. For example:

```
$acl = set-AzDataLakeGen2ItemAclObject -AccessControlType user -Permission rwx -DefaultScope .
```

To see an example that sets ACLs recursively in batches by specifying a batch size, see the [Set-AzDataLakeGen2AclRecursive](#) reference article.

## Update ACLs

When you *update* an ACL, you modify the ACL instead of replacing the ACL. For example, you can add a new security principal to the ACL without affecting other security principals listed in the ACL. To replace the ACL instead of update it, see the [Set ACLs](#) section of this article.

This section shows you how to:

- Update an ACL
- Update ACLs recursively

### Update an ACL

First, get the ACL. Then, use the `Set-AzDataLakeGen2ItemAclObject` cmdlet to add or update an ACL entry. Use the `Update-AzDataLakeGen2Item` cmdlet to commit the ACL.

This example creates or updates the ACL on a directory for a user.

```

$filesystemName = "my-file-system"
$dirname = "my-directory/"
$acl = (Get-AzDataLakeGen2Item -Context $ctx -FileSystem $filesystemName -Path $dirname).ACL
$acl = set-AzDataLakeGen2ItemAclObject -AccessControlType user -EntityID xxxxxxxx-xxxx-xxxxxxxxxx -
Permission r-x -InputObject $acl
Update-AzDataLakeGen2Item -Context $ctx -FileSystem $filesystemName -Path $dirname -Acl $acl

```

#### NOTE

If you want to update a **default** ACL entry, use the **-DefaultScope** parameter when you run the **Set-AzDataLakeGen2ItemAclObject** command. For example:

```
$acl = set-AzDataLakeGen2ItemAclObject -AccessControlType user -EntityID xxxxxxxx-xxxx-xxxxxxxxxx -
Permission r-x -DefaultScope .
```

### Update ACLs recursively

Update ACLs recursively by using the `Update-AzDataLakeGen2AclRecursive` cmdlet.

This example updates an ACL entry with write permission.

```
$filesystemName = "my-container"
$dirname = "my-parent-directory/"
$userID = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx";

$acl = Set-AzDataLakeGen2ItemAclObject -AccessControlType user -EntityId $userID -Permission rwx

Update-AzDataLakeGen2AclRecursive -Context $ctx -FileSystem $filesystemName -Path $dirname -Acl $acl
```

#### NOTE

To set the ACL of a specific group or user, use their respective object IDs. For example,

```
group:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx or user:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx .
```

To see an example that updates ACLs recursively in batches by specifying a batch size, see the [Update-AzDataLakeGen2AclRecursive](#) reference article.

## Remove ACL entries

This section shows you how to:

- Remove an ACL entry
- Remove ACL entries recursively

### Remove an ACL entry

This example removes an entry from an existing ACL.

```
$id = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"

Create the new ACL object.
[Collections.Generic.List[System.Object]]]$aclnew = $acl

foreach ($a in $aclnew)
{
 if ($a.AccessControlType -eq "User"-and $a.DefaultScope -eq $false -and $a.EntityId -eq $id)
 {
 $aclnew.Remove($a);
 break;
 }
}
Update-AzDataLakeGen2Item -Context $ctx -FileSystem $filesystemName -Path $dirname -Acl $aclnew
```

### Remove ACL entries recursively

You can remove one or more ACL entries recursively. To remove an ACL entry, create a new ACL object for ACL entry to be removed, and then use that object in remove ACL operation. Do not get the existing ACL, just provide the ACL entries to be removed.

Remove ACL entries by using the [Remove-AzDataLakeGen2AclRecursive](#) cmdlet.

This example removes an ACL entry from the root directory of the container.

```
$filesystemName = "my-container"
$userID = "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"

$acl = Set-AzDataLakeGen2ItemAclObject -AccessControlType user -EntityId $userID -Permission "---"

Remove-AzDataLakeGen2AclRecursive -Context $ctx -FileSystem $filesystemName -Acl $acl
```

#### NOTE

If you want to remove a **default** ACL entry, use the **-DefaultScope** parameter when you run the **Set-AzDataLakeGen2ItemAclObject** command. For example:

```
$acl = set-AzDataLakeGen2ItemAclObject -AccessControlType user -EntityId $userID -Permission "---" -
DefaultScope
```

To see an example that removes ACLs recursively in batches by specifying a batch size, see the [Remove-AzDataLakeGen2AclRecursive](#) reference article.

## Recover from failures

You might encounter runtime or permission errors when modifying ACLs recursively.

For runtime errors, restart the process from the beginning. Permission errors can occur if the security principal doesn't have sufficient permission to modify the ACL of a directory or file that is in the directory hierarchy being modified. Address the permission issue, and then choose to either resume the process from the point of failure by using a continuation token, or restart the process from beginning. You don't have to use the continuation token if you prefer to restart from the beginning. You can reapply ACL entries without any negative impact.

This example return results to the variable, and then pipes failed entries to a formatted table.

```
$result = Set-AzDataLakeGen2AclRecursive -Context $ctx -FileSystem $filesystemName -Path $dirname -Acl $acl
$result
$result.FailedEntries | ft
```

Based on the output of the table, you can fix any permission errors, and then resume execution by using the continuation token.

```
$result = Set-AzDataLakeGen2AclRecursive -Context $ctx -FileSystem $filesystemName -Path $dirname -Acl $acl
-ContinuationToken $result.ContinuationToken
$result
```

To see an example that sets ACLs recursively in batches by specifying a batch size, see the [Set-AzDataLakeGen2AclRecursive](#) reference article.

If you want the process to complete uninterrupted by permission errors, you can specify that.

This example uses the `ContinueOnFailure` parameter so that execution continues even if the operation encounters a permission error.

```
$result = Set-AzDataLakeGen2AclRecursive -Context $ctx -FileSystem $filesystemName -Path $dirname -Acl $acl
-ContinueOnFailure

echo "[Result Summary]"
echo "TotalDirectoriesSuccessfulCount: `t$(($result.TotalFilesSuccessfulCount))"
echo "TotalFilesSuccessfulCount: `t`t$(($result.TotalDirectoriesSuccessfulCount))"
echo "TotalFailureCount: `t`t`t`t$(($result.TotalFailureCount))"
echo "FailedEntries:"$(($result.FailedEntries | ft)
```

To see an example that sets ACLs recursively in batches by specifying a batch size, see the [Set-AzDataLakeGen2AclRecursive](#) reference article.

## Best practices

This section provides you some best practice guidelines for setting ACLs recursively.

### Handling runtime errors

A runtime error can occur for many reasons (For example: an outage or a client connectivity issue). If you encounter a runtime error, restart the recursive ACL process. ACLs can be reapplied to items without causing a negative impact.

### Handling permission errors (403)

If you encounter an access control exception while running a recursive ACL process, your AD [security principal](#) might not have sufficient permission to apply an ACL to one or more of the child items in the directory hierarchy. When a permission error occurs, the process stops and a continuation token is provided. Fix the permission issue, and then use the continuation token to process the remaining dataset. The directories and files that have already been successfully processed won't have to be processed again. You can also choose to restart the recursive ACL process. ACLs can be reapplied to items without causing a negative impact.

### Credentials

We recommend that you provision an Azure AD security principal that has been assigned the [Storage Blob Data Owner](#) role in the scope of the target storage account or container.

### Performance

To reduce latency, we recommend that you run the recursive ACL process in an Azure Virtual Machine (VM) that is located in the same region as your storage account.

### ACL limits

The maximum number of ACLs that you can apply to a directory or file is 32 access ACLs and 32 default ACLs. For more information, see [Access control in Azure Data Lake Storage Gen2](#).

## See also

- [Known issues](#)
- [Storage PowerShell cmdlets](#)
- [Access control model in Azure Data Lake Storage Gen2](#)
- [Access control lists \(ACLs\) in Azure Data Lake Storage Gen2](#)

# Use Azure CLI to manage ACLs in Azure Data Lake Storage Gen2

8/22/2022 • 9 minutes to read • [Edit Online](#)

This article shows you how to use the [Azure CLI](#) to get, set, and update the access control lists of directories and files.

ACL inheritance is already available for new child items that are created under a parent directory. But you can also add, update, and remove ACLs recursively on the existing child items of a parent directory without having to make these changes individually for each child item.

[Reference](#) | [Samples](#) | [Give feedback](#)

## Prerequisites

- An Azure subscription. For more information, see [Get Azure free trial](#).
- A storage account that has hierarchical namespace enabled. Follow [these](#) instructions to create one.
- Azure CLI version `2.14.0` or higher.
- One of the following security permissions:
  - A provisioned Azure Active Directory (Azure AD) [security principal](#) that has been assigned the [Storage Blob Data Owner](#) role in the scope of either the target container, parent resource group or subscription.
  - Owning user of the target container or directory to which you plan to apply ACL settings. To set ACLs recursively, this includes all child items in the target container or directory.
  - Storage account key.

## Ensure that you have the correct version of Azure CLI installed

1. Open the [Azure Cloud Shell](#), or if you've [installed](#) the Azure CLI locally, open a command console application such as Windows PowerShell.
2. Verify that the version of Azure CLI that have installed is `2.14.0` or higher by using the following command.

```
az --version
```

If your version of Azure CLI is lower than `2.14.0`, then install a later version. For more information, see [Install the Azure CLI](#).

## Connect to the account

1. If you're using Azure CLI locally, run the login command.

```
az login
```

If the CLI can open your default browser, it will do so and load an Azure sign-in page.

Otherwise, open a browser page at <https://aka.ms/devicelogin> and enter the authorization code displayed in your terminal. Then, sign in with your account credentials in the browser.

To learn more about different authentication methods, see [Authorize access to blob or queue data with Azure CLI](#).

2. If your identity is associated with more than one subscription, then set your active subscription to subscription of the storage account that will host your static website.

```
az account set --subscription <subscription-id>
```

Replace the `<subscription-id>` placeholder value with the ID of your subscription.

#### NOTE

The example presented in this article show Azure Active Directory (Azure AD) authorization. To learn more about authorization methods, see [Authorize access to blob or queue data with Azure CLI](#).

## Get ACLs

Get the ACL of a **directory** by using the `az storage fs access show` command.

This example gets the ACL of a directory, and then prints the ACL to the console.

```
az storage fs access show -p my-directory -f my-file-system --account-name mystorageaccount --auth-mode login
```

Get the access permissions of a **file** by using the `az storage fs access show` command.

This example gets the ACL of a file and then prints the ACL to the console.

```
az storage fs access show -p my-directory/upload.txt -f my-file-system --account-name mystorageaccount --auth-mode login
```

The following image shows the output after getting the ACL of a directory.

```
{
 "acl": "user::rwx,group::r-x,other::---",
 "group": "$superuser",
 "owner": "44444444-4444-4444-4444-444444444444",
 "permissions": "rwxr-x---"
}
```

In this example, the owning user has read, write, and execute permissions. The owning group has only read and execute permissions. For more information about access control lists, see [Access control in Azure Data Lake Storage Gen2](#).

## Set ACLs

When you *set* an ACL, you *replace* the entire ACL including all of its entries. If you want to change the permission level of a security principal or add a new security principal to the ACL without affecting other existing entries, you should *update* the ACL instead. To update an ACL instead of replace it, see the [Update ACLs](#) section of this article.

If you choose to *set* the ACL, you must add an entry for the owning user, an entry for the owning group, and an

entry for all other users. To learn more about the owning user, the owning group, and all other users, see [Users and identities](#).

This section shows you how to:

- Set an ACL
- Set ACLs recursively

## Set an ACL

Use the [az storage fs access set](#) command to set the ACL of a **directory**.

This example sets the ACL on a directory for the owning user, owning group, or other users, and then prints the ACL to the console.

```
az storage fs access set --acl "user::rw-,group::rw-,other::wx" -p my-directory -f my-file-system --account-name mystorageaccount --auth-mode login
```

This example sets the *default* ACL on a directory for the owning user, owning group, or other users, and then prints the ACL to the console.

```
az storage fs access set --acl "default:user::rw-,group::rw-,other::wx" -p my-directory -f my-file-system --account-name mystorageaccount --auth-mode login
```

Use the [az storage fs access set](#) command to set the acl of a **file**.

This example sets the ACL on a file for the owning user, owning group, or other users, and then prints the ACL to the console.

```
az storage fs access set --acl "user::rw-,group::rw-,other::wx" -p my-directory/upload.txt -f my-file-system --account-name mystorageaccount --auth-mode login
```

### NOTE

To set the ACL of a specific group or user, use their respective object IDs. For example,

`group:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx` or `user:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`.

The following image shows the output after setting the ACL of a file.

```
{
 "acl": "user::rw-,group::rw-,other::wx",
 "group": "$superuser",
 "owner": "$superuser",
 "permissions": "rw-rw--wx"
}
```

In this example, the owning user and owning group have only read and write permissions. All other users have write and execute permissions. For more information about access control lists, see [Access control in Azure Data Lake Storage Gen2](#).

## Set ACLs recursively

Set ACLs recursively by using the [az storage fs access set-recursive](#) command.

This example sets the ACL of a directory named `my-parent-directory`. These entries give the owning user read, write, and execute permissions, gives the owning group only read and execute permissions, and gives all others no access. The last ACL entry in this example gives a specific user with the object ID "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx" read and execute permissions.

```
az storage fs access set-recursive --acl "user::rwx,group::r-x,other::---,user:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx:r-x" -p my-parent-directory/ -f my-container --account-name mystorageaccount --auth-mode login
```

#### NOTE

If you want to set a **default** ACL entry, add the prefix `default:` to each entry. For example, `default:user::rwx` or `default:user:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx:r-x`.

## Update ACLs

When you *update* an ACL, you modify the ACL instead of replacing the ACL. For example, you can add a new security principal to the ACL without affecting other security principals listed in the ACL. To replace the ACL instead of update it, see the [Set ACLs](#) section of this article.

To update an ACL, create a new ACL object with the ACL entry that you want to update, and then use that object in update ACL operation. Do not get the existing ACL, just provide ACL entries to be updated.

This section shows you how to:

- Update an ACL
- Update ACLs recursively

### Update an ACL

Another way to set this permission is to use the [az storage fs access set](#) command.

Update the ACL of a directory or file by setting the `-permissions` parameter to the short form of an ACL.

This example updates the ACL of a **directory**.

```
az storage fs access set --permissions rwxrwxrwx -p my-directory -f my-file-system --account-name mystorageaccount --auth-mode login
```

This example updates the ACL of a **file**.

```
az storage fs access set --permissions rwxrwxrwx -p my-directory/upload.txt -f my-file-system --account-name mystorageaccount --auth-mode login
```

#### NOTE

To update the ACL of a specific group or user, use their respective object IDs. For example,

```
group:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx or user:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx .
```

You can also update the owning user and group of a directory or file by setting the `--owner` or `group` parameters to the entity ID or User Principal Name (UPN) of a user.

This example changes the owner of a directory.

```
az storage fs access set --owner xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx -p my-directory -f my-file-system --account-name mystorageaccount --auth-mode login
```

This example changes the owner of a file.

```
az storage fs access set --owner xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx -p my-directory/upload.txt -f my-file-system --account-name mystorageaccount --auth-mode login
```

## Update ACLs recursively

Update ACLs recursively by using the [az storage fs access update-recursive](#) command.

This example updates an ACL entry with write permission.

```
az storage fs access update-recursive --acl "user:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx:rwx" -p my-parent-directory/ -f my-container --account-name mystorageaccount --auth-mode login
```

### NOTE

If you want to update a **default** ACL entry, add the prefix `default:` to each entry. For example,  
`default:user:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx:r-x`.

## Remove ACL entries recursively

You can remove one or more ACL entries recursively. To remove an ACL entry, create a new ACL object for ACL entry to be removed, and then use that object in remove ACL operation. Do not get the existing ACL, just provide the ACL entries to be removed.

Remove ACL entries by using the [az storage fs access remove-recursive](#) command.

This example removes an ACL entry from the root directory of the container.

```
az storage fs access remove-recursive --acl "user:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx" -p my-parent-directory/ -f my-container --account-name mystorageaccount --auth-mode login
```

### NOTE

If you want to remove a **default** ACL entry, add the prefix `default:` to each entry. For example,  
`default:user:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`.

## Recover from failures

You might encounter runtime or permission errors when modifying ACLs recursively. For runtime errors, restart the process from the beginning. Permission errors can occur if the security principal doesn't have sufficient permission to modify the ACL of a directory or file that is in the directory hierarchy being modified. Address the permission issue, and then choose to either resume the process from the point of failure by using a continuation token, or restart the process from beginning. You don't have to use the continuation token if you prefer to restart from the beginning. You can reapply ACL entries without any negative impact.

In the event of a failure, you can return a continuation token by setting the `--continue-on-failure` parameter to `false`. After you address the errors, you can resume the process from the point of failure by running the command again, and then setting the `--continuation` parameter to the continuation token.

```
az storage fs access set-recursive --acl "user::rw-,group::r-x,other::---" --continue-on-failure false --continuation xxxxxxx -p my-parent-directory/ -f my-container --account-name mystorageaccount --auth-mode login
```

If you want the process to complete uninterrupted by permission errors, you can specify that.

To ensure that the process completes uninterrupted, set the `--continue-on-failure` parameter to `true`.

```
az storage fs access set-recurse --acl "user::rw-,group::r-x,other::---" --continue-on-failure true --continuation xxxxxxxx -p my-parent-directory/ -f my-container --account-name mystorageaccount --auth-mode login
```

## Best practices

This section provides you some best practice guidelines for setting ACLs recursively.

### Handling runtime errors

A runtime error can occur for many reasons (For example: an outage or a client connectivity issue). If you encounter a runtime error, restart the recursive ACL process. ACLs can be reapplied to items without causing a negative impact.

### Handling permission errors (403)

If you encounter an access control exception while running a recursive ACL process, your AD [security principal](#) might not have sufficient permission to apply an ACL to one or more of the child items in the directory hierarchy. When a permission error occurs, the process stops and a continuation token is provided. Fix the permission issue, and then use the continuation token to process the remaining dataset. The directories and files that have already been successfully processed won't have to be processed again. You can also choose to restart the recursive ACL process. ACLs can be reapplied to items without causing a negative impact.

### Credentials

We recommend that you provision an Azure AD security principal that has been assigned the [Storage Blob Data Owner](#) role in the scope of the target storage account or container.

### Performance

To reduce latency, we recommend that you run the recursive ACL process in an Azure Virtual Machine (VM) that is located in the same region as your storage account.

### ACL limits

The maximum number of ACLs that you can apply to a directory or file is 32 access ACLs and 32 default ACLs. For more information, see [Access control in Azure Data Lake Storage Gen2](#).

## See also

- [Samples](#)
- [Give feedback](#)
- [Known issues](#)
- [Access control model in Azure Data Lake Storage Gen2](#)
- [Access control lists \(ACLs\) in Azure Data Lake Storage Gen2](#)

# Use .NET to manage ACLs in Azure Data Lake Storage Gen2

8/22/2022 • 11 minutes to read • [Edit Online](#)

This article shows you how to use .NET to get, set, and update the access control lists of directories and files.

ACL inheritance is already available for new child items that are created under a parent directory. But you can also add, update, and remove ACLs recursively on the existing child items of a parent directory without having to make these changes individually for each child item.

[Package \(NuGet\)](#) | [Samples](#) | [API reference](#) | [Gen1 to Gen2 mapping](#) | [Give Feedback](#)

## Prerequisites

- An Azure subscription. See [Get Azure free trial](#).
- A storage account that has hierarchical namespace (HNS) enabled. Follow [these](#) instructions to create one.
- Azure CLI version `2.6.0` or higher.
- One of the following security permissions:
  - A provisioned Azure Active Directory (AD) [security principal](#) that has been assigned the [Storage Blob Data Owner](#) role in the scope of either the target container, parent resource group or subscription.
  - Owning user of the target container or directory to which you plan to apply ACL settings. To set ACLs recursively, this includes all child items in the target container or directory.
  - Storage account key.

## Set up your project

To get started, install the [Azure.Storage.Files.DataLake](#) NuGet package.

1. Open a command window (For example: Windows PowerShell).
2. From your project directory, install the [Azure.Storage.Files.DataLake](#) preview package by using the `dotnet add package` command.

```
dotnet add package Azure.Storage.Files.DataLake -v 12.6.0 -s https://pkgs.dev.azure.com/azure-sdk/public/_packaging/azure-sdk-for-net/nuget/v3/index.json
```

Then, add these using statements to the top of your code file.

```
using Azure;
using Azure.Core;
using Azure.Storage;
using Azure.Storage.Files.DataLake;
using Azure.Storage.Files.DataLake.Models;
using System.Collections.Generic;
using System.Threading.Tasks;
```

# Connect to the account

To use the snippets in this article, you'll need to create a [DataLakeServiceClient](#) instance that represents the storage account.

## Connect by using Azure Active Directory (AD)

### NOTE

If you're using Azure Active Directory (Azure AD) to authorize access, then make sure that your security principal has been assigned the [Storage Blob Data Owner role](#). To learn more about how ACL permissions are applied and the effects of changing them, see [Access control model in Azure Data Lake Storage Gen2](#).

You can use the [Azure identity client library for .NET](#) to authenticate your application with Azure AD.

After you install the package, add this using statement to the top of your code file.

```
using Azure.Identity;
```

Get a client ID, a client secret, and a tenant ID. To do this, see [Acquire a token from Azure AD for authorizing requests from a client application](#). As part of that process, you'll have to assign one of the following [Azure role-based access control \(Azure RBAC\)](#) roles to your security principal.

ROLE	ACL SETTING CAPABILITY
Storage Blob Data Owner	All directories and files in the account.
Storage Blob Data Contributor	Only directories and files owned by the security principal.

This example creates a [DataLakeServiceClient](#) instance by using a client ID, a client secret, and a tenant ID.

```
public static void GetDataLakeServiceClient(ref DataLakeServiceClient dataLakeServiceClient,
 String accountName, String clientID, string clientSecret, string tenantID)
{
 TokenCredential credential = new ClientSecretCredential(
 tenantID, clientID, clientSecret, new TokenCredentialOptions());
 string dfsUri = "https://" + accountName + ".dfs.core.windows.net";
 dataLakeServiceClient = new DataLakeServiceClient(new Uri(dfsUri), credential);
}
```

### NOTE

For more examples, see the [Azure identity client library for .NET](#) documentation..

## Connect by using an account key

This is the easiest way to connect to an account.

This example creates a [DataLakeServiceClient](#) instance by using an account key.

```

public static void GetDataLakeServiceClient(ref DataLakeServiceClient dataLakeServiceClient,
 string accountName, string accountKey)
{
 StorageSharedKeyCredential sharedKeyCredential =
 new StorageSharedKeyCredential(accountName, accountKey);

 string dfsUri = "https://" + accountName + ".dfs.core.windows.net";

 dataLakeServiceClient = new DataLakeServiceClient
 (new Uri(dfsUri), sharedKeyCredential);
}

```

## Set ACLs

When you *set* an ACL, you *replace* the entire ACL including all of its entries. If you want to change the permission level of a security principal or add a new security principal to the ACL without affecting other existing entries, you should *update* the ACL instead. To update an ACL instead of replace it, see the [Update ACLs](#) section of this article.

If you choose to *set* the ACL, you must add an entry for the owning user, an entry for the owning group, and an entry for all other users. To learn more about the owning user, the owning group, and all other users, see [Users and identities](#).

This section shows you how to:

- Set the ACL of a directory
- Set the ACL of a file
- Set ACLs recursively

### Set the ACL of a directory

Get the access control list (ACL) of a directory by calling the [DataLakeDirectoryClient.GetAccessControlAsync](#) method and set the ACL by calling the [DataLakeDirectoryClient.SetAccessControlList](#) method.

This example gets and sets the ACL of a directory named `my-directory`. The string

`user::rwx,group::r-x,other::rw-` gives the owning user read, write, and execute permissions, gives the owning group only read and execute permissions, and gives all others read and write permission.

```

public async Task ManageDirectoryACLs(DataLakeFileSystemClient fileSystemClient)
{
 DataLakeDirectoryClient directoryClient =
 fileSystemClient.GetDirectoryClient("");

 PathAccessControl directoryAccessControl =
 await directoryClient.GetAccessControlAsync();

 foreach (var item in directoryAccessControl.AccessControlList)
 {
 Console.WriteLine(item.ToString());
 }

 IList<PathAccessControlItem> accessControlList
 = PathAccessControlExtensions.ParseAccessControlList
 ("user::rwx,group::r-x,other::rw-");

 directoryClient.SetAccessControlList(accessControlList);
}

```

You can also get and set the ACL of the root directory of a container. To get the root directory, pass an empty

string ( `""` ) into the [DataLakeFileSystemClient.GetDirectoryClient](#) method.

## Set the ACL of a file

Get the access control list (ACL) of a file by calling the [DataLakeFileClient.GetAccessControlAsync](#) method and set the ACL by calling the [DataLakeFileClient.SetAccessControlList](#) method.

This example gets and sets the ACL of a file named `my-file.txt`. The string `user::rwx,group::r-x,other::rw-` gives the owning user read, write, and execute permissions, gives the owning group only read and execute permissions, and gives all others read and write permission.

```
public async Task ManageFileACls(DataLakeFileSystemClient fileSystemClient)
{
 DataLakeDirectoryClient directoryClient =
 fileSystemClient.GetDirectoryClient("my-directory");

 DataLakeFileClient fileClient =
 directoryClient.GetFileClient("hello.txt");

 PathAccessControl FileAccessControl =
 await fileClient.GetAccessControlAsync();

 foreach (var item in FileAccessControl.AccessControlList)
 {
 Console.WriteLine(item.ToString());
 }

 IList<PathAccessControlItem> accessControlList
 = PathAccessControlExtensions.ParseAccessControlList
 ("user::rwx,group::r-x,other::rw-");

 fileClient.SetAccessControlList(accessControlList);
}
```

## Set ACLs recursively

Set ACLs recursively by calling the [DataLakeDirectoryClient.SetAccessControlRecursiveAsync](#) method. Pass this method a [List](#) of [PathAccessControlItem](#). Each [PathAccessControlItem](#) defines an ACL entry.

If you want to set a **default** ACL entry, then you can set the [PathAccessControlItem.DefaultScope](#) property of the [PathAccessControlItem](#) to **true**.

This example sets the ACL of a directory named `my-parent-directory`. This method accepts a boolean parameter named `isDefaultScope` that specifies whether to set the default ACL. That parameter is used in the constructor of the [PathAccessControlItem](#). The entries of the ACL give the owning user read, write, and execute permissions, gives the owning group only read and execute permissions, and gives all others no access. The last ACL entry in this example gives a specific user with the object ID `"xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"` read and execute permissions.

```

public async Task SetACLRecursively(DataLakeServiceClient serviceClient, bool isDefaultScope)
{
 DataLakeDirectoryClient directoryClient =
 serviceClient.GetFileSystemClient("my-container").
 GetDirectoryClient("my-parent-directory");

 List<PathAccessControlItem> accessControlList =
 new List<PathAccessControlItem>()
 {
 new PathAccessControlItem(AccessControlType.User,
 RolePermissions.Read |
 RolePermissions.Write |
 RolePermissions.Execute, isDefaultScope),

 new PathAccessControlItem(AccessControlType.Group,
 RolePermissions.Read |
 RolePermissions.Execute, isDefaultScope),

 new PathAccessControlItem(AccessControlType.Other,
 RolePermissions.None, isDefaultScope),

 new PathAccessControlItem(AccessControlType.User,
 RolePermissions.Read |
 RolePermissions.Execute, isDefaultScope,
 entityId: "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"),
 };

 await directoryClient.SetAccessControlRecursiveAsync
 (accessControlList, null);
}

```

## Update ACLs

When you *update* an ACL, you modify the ACL instead of replacing the ACL. For example, you can add a new security principal to the ACL without affecting other security principals listed in the ACL. To replace the ACL instead of update it, see the [Set ACLs](#) section of this article.

This section shows you how to:

- Update an ACL
- Update ACLs recursively

### Update an ACL

First, get the ACL of a directory by calling the [DataLakeDirectoryClient.GetAccessControlAsync](#) method. Copy the list of ACL entries to a new [List](#) of [PathAccessControl](#) objects. Then locate the entry that you want to update and replace it in the list. Set the ACL by calling the [DataLakeDirectoryClient.SetAccessControlList](#) method.

This example updates the root ACL of a container by replacing the ACL entry for all other users.

```

public async Task UpdateDirectoryACls(DataLakeFileSystemClient fileSystemClient)
{
 DataLakeDirectoryClient directoryClient =
 fileSystemClient.GetDirectoryClient("");

 PathAccessControl directoryAccessControl =
 await directoryClient.GetAccessControlAsync();

 List<PathAccessControlItem> accessControlListUpdate
 = (List<PathAccessControlItem>)directoryAccessControl.AccessControlList;

 int index = -1;

 foreach (var item in accessControlListUpdate)
 {
 if (item.AccessControlType == AccessControlType.Other)
 {
 index = accessControlListUpdate.IndexOf(item);
 break;
 }
 }

 if (index > -1)
 {
 accessControlListUpdate[index] = new PathAccessControlItem(AccessControlType.Other,
 RolePermissions.Read |
 RolePermissions.Execute);

 directoryClient.SetAccessControlList(accessControlListUpdate);
 }
}

```

## Update ACLs recursively

To update an ACL recursively, create a new ACL object with the ACL entry that you want to update, and then use that object in update ACL operation. Do not get the existing ACL, just provide ACL entries to be updated.

Update an ACL recursively by calling the `DataLakeDirectoryClient.UpdateAccessControlRecursiveAsync` method. Pass this method a [List](#) of `PathAccessControlItem`. Each `PathAccessControlItem` defines an ACL entry.

If you want to update a **default** ACL entry, then you can set the `PathAccessControlItem.DefaultScope` property of the `PathAccessControlItem` to `true`.

This example updates an ACL entry with write permission. This method accepts a boolean parameter named `isDefaultScope` that specifies whether to update the default ACL. That parameter is used in the constructor of the `PathAccessControlItem`.

```
public async Task UpdateACLSRecursively(DataLakeServiceClient serviceClient, bool isDefaultScope)
{
 DataLakeDirectoryClient directoryClient =
 serviceClient.GetFileSystemClient("my-container").
 GetDirectoryClient("my-parent-directory");

 List<PathAccessControlItem> accessControlListUpdate =
 new List<PathAccessControlItem>()
 {
 new PathAccessControlItem(AccessControlType.User,
 RolePermissions.Read |
 RolePermissions.Write |
 RolePermissions.Execute, isDefaultScope,
 entityId: "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"),
 };

 await directoryClient.UpdateAccessControlRecursiveAsync
 (accessControlListUpdate, null);
}
```

## Remove ACL entries

You can remove one or more ACL entries. This section shows you how to:

- Remove an ACL entry
- Remove ACL entries recursively

### Remove an ACL entry

First, get the ACL of a directory by calling the [DataLakeDirectoryClient.GetAccessControlAsync](#) method. Copy the list of ACL entries to a new [List](#) of [PathAccessControl](#) objects. Then locate the entry that you want to remove and call the [Remove](#) method of the collection. Set the updated ACL by calling the [DataLakeDirectoryClient.SetAccessControlList](#) method.

This example updates the root ACL of a container by replacing the ACL entry for all other users.

```

public async Task RemoveDirectoryACLEntry
 (DataLakeFileSystemClient fileSystemClient)
{
 DataLakeDirectoryClient directoryClient =
 fileSystemClient.GetDirectoryClient("");

 PathAccessControl directoryAccessControl =
 await directoryClient.GetAccessControlAsync();

 List<PathAccessControlItem> accessControlListUpdate
 = (List<PathAccessControlItem>)directoryAccessControl.AccessControlList;

 PathAccessControlItem entryToRemove = null;

 foreach (var item in accessControlListUpdate)
 {
 if (item.EntityId == "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx")
 {
 entryToRemove = item;
 break;
 }
 }

 if (entryToRemove != null)
 {
 accessControlListUpdate.Remove(entryToRemove);
 directoryClient.SetAccessControlList(accessControlListUpdate);
 }
}

```

## Remove ACL entries recursively

To remove ACL entries recursively, create a new ACL object for ACL entry to be removed, and then use that object in remove ACL operation. Do not get the existing ACL, just provide the ACL entries to be removed.

Remove ACL entries by calling the `DataLakeDirectoryClient.RemoveAccessControlRecursiveAsync` method. Pass this method a [List](#) of `PathAccessControlItem`. Each `PathAccessControlItem` defines an ACL entry.

If you want to remove a **default** ACL entry, then you can set the `PathAccessControlItem.DefaultScope` property of the `PathAccessControlItem` to **true**.

This example removes an ACL entry from the ACL of the directory named `my-parent-directory`. This method accepts a boolean parameter named `isDefaultScope` that specifies whether to remove the entry from the default ACL. That parameter is used in the constructor of the `PathAccessControlItem`.

```

public async Task RemoveACLSRecursively(DataLakeServiceClient serviceClient, bool isDefaultScope)
{
 DataLakeDirectoryClient directoryClient =
 serviceClient.GetFileSystemClient("my-container").
 GetDirectoryClient("my-parent-directory");

 List<RemovePathAccessControlItem> accessControlListForRemoval =
 new List<RemovePathAccessControlItem>()
 {
 new RemovePathAccessControlItem(AccessControlType.User, isDefaultScope,
 entityId: "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"),
 };

 await directoryClient.RemoveAccessControlRecursiveAsync
 (accessControlListForRemoval, null);
}

```

## Recover from failures

You might encounter runtime or permission errors when modifying ACLs recursively. For runtime errors, restart the process from the beginning. Permission errors can occur if the security principal doesn't have sufficient permission to modify the ACL of a directory or file that is in the directory hierarchy being modified. Address the permission issue, and then choose to either resume the process from the point of failure by using a continuation token, or restart the process from beginning. You don't have to use the continuation token if you prefer to restart from the beginning. You can reapply ACL entries without any negative impact.

This example returns a continuation token in the event of a failure. The application can call this example method again after the error has been addressed, and pass in the continuation token. If this example method is called for the first time, the application can pass in a value of `null` for the continuation token parameter.

```
public async Task<string> ResumeAsync(DataLakeServiceClient serviceClient,
 DataLakeDirectoryClient directoryClient,
 List<PathAccessControlItem> accessControlList,
 string continuationToken)
{
 try
 {
 var accessControlChangeResult =
 await directoryClient.SetAccessControlRecursiveAsync(
 accessControlList, continuationToken: continuationToken, null);

 if (accessControlChangeResult.Value.CountersFailedChangesCount > 0)
 {
 continuationToken =
 accessControlChangeResult.Value.ContinuationToken;
 }

 return continuationToken;
 }
 catch (Exception ex)
 {
 Console.WriteLine(ex.ToString());
 return continuationToken;
 }
}
```

If you want the process to complete uninterrupted by permission errors, you can specify that.

To ensure that the process completes uninterrupted, pass in an `AccessControlChangedOptions` object and set the `ContinueOnFailure` property of that object to `true`.

This example sets ACL entries recursively. If this code encounters a permission error, it records that failure and continues execution. This example prints the number of failures to the console.

```

public async Task ContinueOnFailureAsync(DataLakeServiceClient serviceClient,
 DataLakeDirectoryClient directoryClient,
 List<PathAccessControlItem> accessControlList)
{
 var accessControlChangeResult =
 await directoryClient.SetAccessControlRecursiveAsync(
 accessControlList, null, new AccessControlChangeOptions()
 { ContinueOnFailure = true });

 var counters = accessControlChangeResult.Value.Counters;

 Console.WriteLine("Number of directories changed: " +
 counters.ChangedDirectoriesCount.ToString());

 Console.WriteLine("Number of files changed: " +
 counters.ChangedFilesCount.ToString());

 Console.WriteLine("Number of failures: " +
 counters.FailedChangesCount.ToString());
}

```

## Best practices

This section provides you some best practice guidelines for setting ACLs recursively.

### **Handling runtime errors**

A runtime error can occur for many reasons (For example: an outage or a client connectivity issue). If you encounter a runtime error, restart the recursive ACL process. ACLs can be reapplied to items without causing a negative impact.

### **Handling permission errors (403)**

If you encounter an access control exception while running a recursive ACL process, your AD [security principal](#) might not have sufficient permission to apply an ACL to one or more of the child items in the directory hierarchy. When a permission error occurs, the process stops and a continuation token is provided. Fix the permission issue, and then use the continuation token to process the remaining dataset. The directories and files that have already been successfully processed won't have to be processed again. You can also choose to restart the recursive ACL process. ACLs can be reapplied to items without causing a negative impact.

### **Credentials**

We recommend that you provision an Azure AD security principal that has been assigned the [Storage Blob Data Owner](#) role in the scope of the target storage account or container.

### **Performance**

To reduce latency, we recommend that you run the recursive ACL process in an Azure Virtual Machine (VM) that is located in the same region as your storage account.

### **ACL limits**

The maximum number of ACLs that you can apply to a directory or file is 32 access ACLs and 32 default ACLs. For more information, see [Access control in Azure Data Lake Storage Gen2](#).

## See also

- [API reference documentation](#)
- [Package \(NuGet\)](#)
- [Samples](#)
- [Gen1 to Gen2 mapping](#)
- [Known issues](#)
- [Give Feedback](#)

- Access control model in Azure Data Lake Storage Gen2
- Access control lists (ACLs) in Azure Data Lake Storage Gen2

# Use Java to manage ACLs in Azure Data Lake Storage Gen2

8/22/2022 • 12 minutes to read • [Edit Online](#)

This article shows you how to use Java to get, set, and update the access control lists of directories and files.

ACL inheritance is already available for new child items that are created under a parent directory. But you can also add, update, and remove ACLs recursively on the existing child items of a parent directory without having to make these changes individually for each child item.

[Package \(Maven\)](#) | [Samples](#) | [API reference](#) | [Gen1 to Gen2 mapping](#) | [Give Feedback](#)

## Prerequisites

- An Azure subscription. See [Get Azure free trial](#).
- A storage account that has hierarchical namespace (HNS) enabled. Follow [these](#) instructions to create one.
- Azure CLI version `2.6.0` or higher.
- One of the following security permissions:
  - A provisioned Azure Active Directory (AD) [security principal](#) that has been assigned the [Storage Blob Data Owner](#) role in the scope of either the target container, parent resource group or subscription.
  - Owning user of the target container or directory to which you plan to apply ACL settings. To set ACLs recursively, this includes all child items in the target container or directory.
  - Storage account key.

## Set up your project

To get started, open [this page](#) and find the latest version of the Java library. Then, open the `pom.xml` file in your text editor. Add a dependency element that references that version.

If you plan to authenticate your client application by using Azure Active Directory (AD), then add a dependency to the Azure Secret Client Library. See [Adding the Secret Client Library package to your project](#).

Next, add these imports statements to your code file.

```

import com.azure.storage.common.StorageSharedKeyCredential;
import com.azure.storage.file.datalake.DataLakeDirectoryClient;
import com.azure.storage.file.datalake.DataLakeFileClient;
import com.azure.storage.file.datalake.DataLakeFileSystemClient;
import com.azure.storage.file.datalake.DataLakeServiceClient;
import com.azure.storage.file.datalake.DataLakeServiceClientBuilder;
import com.azure.storage.file.datalake.models.ListPathsOptions;
import com.azure.storage.file.datalake.models.PathItem;
import com.azure.storage.file.datalake.models.AccessControlChangeCounters;
import com.azure.storage.file.datalake.models.AccessControlChangeResult;
import com.azure.storage.file.datalake.models.AccessControlType;
import com.azure.storage.file.datalake.models.PathAccessControl;
import com.azure.storage.file.datalake.models.PathAccessControlEntry;
import com.azure.storage.file.datalake.models.PathPermissions;
import com.azure.storage.file.datalake.models.PathRemoveAccessControlEntry;
import com.azure.storage.file.datalake.models.RolePermissions;
import com.azure.storage.file.datalake.options.PathSetAccessControlRecursiveOptions;

```

## Connect to the account

To use the snippets in this article, you'll need to create a `DataLakeServiceClient` instance that represents the storage account.

### Connect by using Azure Active Directory (Azure AD)

You can use the [Azure identity client library for Java](#) to authenticate your application with Azure AD.

Get a client ID, a client secret, and a tenant ID. To do this, see [Acquire a token from Azure AD for authorizing requests from a client application](#). As part of that process, you'll have to assign one of the following [Azure role-based access control \(Azure RBAC\)](#) roles to your security principal.

ROLE	ACL SETTING CAPABILITY
Storage Blob Data Owner	All directories and files in the account.
Storage Blob Data Contributor	Only directories and files owned by the security principal.

This example creates a `DataLakeServiceClient` instance by using a client ID, a client secret, and a tenant ID.

```

static public DataLakeServiceClient GetDataLakeServiceClient
 (String accountName, String clientId, String ClientSecret, String tenantID){

 String endpoint = "https://" + accountName + ".dfs.core.windows.net";

 ClientSecretCredential clientSecretCredential = new ClientSecretCredentialBuilder()
 .clientId(clientId)
 .clientSecret(ClientSecret)
 .tenantId(tenantID)
 .build();

 DataLakeServiceClientBuilder builder = new DataLakeServiceClientBuilder();
 return builder.credential(clientSecretCredential).endpoint(endpoint).buildClient();
}

```

#### NOTE

For more examples, see the [Azure identity client library for Java](#) documentation.

### Connect by using an account key

This is the easiest way to connect to an account.

This example creates a **DataLakeServiceClient** instance by using an account key.

```
static public DataLakeServiceClient GetDataLakeServiceClient
(String accountName, String accountKey){

 StorageSharedKeyCredential sharedKeyCredential =
 new StorageSharedKeyCredential(accountName, accountKey);

 DataLakeServiceClientBuilder builder = new DataLakeServiceClientBuilder();

 builder.credential(sharedKeyCredential);
 builder.endpoint("https://" + accountName + ".dfs.core.windows.net");

 return builder.buildClient();
}
```

## Set ACLs

When you *set* an ACL, you *replace* the entire ACL including all of its entries. If you want to change the permission level of a security principal or add a new security principal to the ACL without affecting other existing entries, you should *update* the ACL instead. To update an ACL instead of replace it, see the [Update ACLs](#) section of this article.

If you choose to *set* the ACL, you must add an entry for the owning user, an entry for the owning group, and an entry for all other users. To learn more about the owning user, the owning group, and all other users, see [Users and identities](#).

This section shows you how to:

- Set the ACL of a directory
- Set the ACL of a file
- Set ACLs recursively

### Set the ACL of a directory

This example gets and then sets the ACL of a directory named `my-directory`. This example gives the owning user read, write, and execute permissions, gives the owning group only read and execute permissions, and gives all others read access.

```

public void ManageDirectoryACLS(DataLakeFileSystemClient fileSystemClient){

 DataLakeDirectoryClient directoryClient =
 fileSystemClient.getDirectoryClient("");

 PathAccessControl directoryAccessControl =
 directoryClient.getAccessControl();

 List<PathAccessControlEntry> pathPermissions = directoryAccessControl.getAccessControlList();

 System.out.println(PathAccessControlEntry.serializeList(pathPermissions));

 RolePermissions groupPermission = new RolePermissions();
 groupPermission.setExecutePermission(true).setReadPermission(true);

 RolePermissions ownerPermission = new RolePermissions();
 ownerPermission.setExecutePermission(true).setReadPermission(true).setWritePermission(true);

 RolePermissions otherPermission = new RolePermissions();
 otherPermission.setReadPermission(true);

 PathPermissions permissions = new PathPermissions();

 permissions.setGroup(groupPermission);
 permissions.setOwner(ownerPermission);
 permissions.setOther(otherPermission);

 directoryClient.setPermissions(permissions, null, null);

 pathPermissions = directoryClient.getAccessControl().getAccessControlList();

 System.out.println(PathAccessControlEntry.serializeList(pathPermissions));

}

```

You can also get and set the ACL of the root directory of a container. To get the root directory, pass an empty string ( `""` ) into the `DataLakeFileSystemClient.getDirectoryClient` method.

### **Set the ACL of a file**

This example gets and then sets the ACL of a file named `upload-file.txt`. This example gives the owning user read, write, and execute permissions, gives the owning group only read and execute permissions, and gives all others read access.

```

public void ManageFileACls(DataLakeFileSystemClient fileSystemClient){

 DataLakeDirectoryClient directoryClient =
 fileSystemClient.getDirectoryClient("my-directory");

 DataLakeFileClient fileClient =
 directoryClient.getFileClient("uploaded-file.txt");

 PathAccessControl fileAccessControl =
 fileClient.getAccessControl();

 List<PathAccessControlEntry> pathPermissions = fileAccessControl.getAccessControlList();

 System.out.println(PathAccessControlEntry.serializeList(pathPermissions));

 RolePermissions groupPermission = new RolePermissions();
 groupPermission.setExecutePermission(true).setReadPermission(true);

 RolePermissions ownerPermission = new RolePermissions();
 ownerPermission.setExecutePermission(true).setReadPermission(true).setWritePermission(true);

 RolePermissions otherPermission = new RolePermissions();
 otherPermission.setReadPermission(true);

 PathPermissions permissions = new PathPermissions();

 permissions.setGroup(groupPermission);
 permissions.setOwner(ownerPermission);
 permissions.setOther(otherPermission);

 fileClient.setPermissions(permissions, null, null);

 pathPermissions = fileClient.getAccessControl().getAccessControlList();

 System.out.println(PathAccessControlEntry.serializeList(pathPermissions));

}

```

## Set ACLs recursively

Set ACLs recursively by calling the `DataLakeDirectoryClient.setAccessControlRecursive` method. Pass this method a [List](#) of `PathAccessControlEntry` objects. Each `PathAccessControlEntry` defines an ACL entry.

If you want to set a **default** ACL entry, then you can call the `setDefaultScope` method of the `PathAccessControlEntry` and pass in a value of `true`.

This example sets the ACL of a directory named `my-parent-directory`. This method accepts a boolean parameter named `isDefaultScope` that specifies whether to set the default ACL. That parameter is used in each call to the `setDefaultScope` method of the `PathAccessControlEntry`. The entries of the ACL give the owning user read, write, and execute permissions, gives the owning group only read and execute permissions, and gives all others no access. The last ACL entry in this example gives a specific user with the object ID "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx" read and execute permissions.

```

public void SetACLRecursively(DataLakeFileSystemClient fileSystemClient, Boolean isDefaultScope){

 DataLakeDirectoryClient directoryClient =
 fileSystemClient.getDirectoryClient("my-parent-directory");

 List<PathAccessControlEntry> pathAccessControlEntries =
 new ArrayList<PathAccessControlEntry>();

 // Create owner entry.
 PathAccessControlEntry ownerEntry = new PathAccessControlEntry();

 RolePermissions ownerPermission = new RolePermissions();
 ownerPermission.setExecutePermission(true).setReadPermission(true).setWritePermission(true);

 ownerEntry.setDefaultScope(isDefaultScope);
 ownerEntry.setAccessControlType(AccessControlType.USER);
 ownerEntry.setPermissions(ownerPermission);

 pathAccessControlEntries.add(ownerEntry);

 // Create group entry.
 PathAccessControlEntry groupEntry = new PathAccessControlEntry();

 RolePermissions groupPermission = new RolePermissions();
 groupPermission.setExecutePermission(true).setReadPermission(true).setWritePermission(false);

 groupEntry.setDefaultScope(isDefaultScope);
 groupEntry.setAccessControlType(AccessControlType.GROUP);
 groupEntry.setPermissions(groupPermission);

 pathAccessControlEntries.add(groupEntry);

 // Create other entry.
 PathAccessControlEntry otherEntry = new PathAccessControlEntry();

 RolePermissions otherPermission = new RolePermissions();
 otherPermission.setExecutePermission(false).setReadPermission(false).setWritePermission(false);

 otherEntry.setDefaultScope(isDefaultScope);
 otherEntry.setAccessControlType(AccessControlType.OTHER);
 otherEntry.setPermissions(otherPermission);

 pathAccessControlEntries.add(otherEntry);

 // Create named user entry.
 PathAccessControlEntry userEntry = new PathAccessControlEntry();

 RolePermissions userPermission = new RolePermissions();
 userPermission.setExecutePermission(true).setReadPermission(true).setWritePermission(false);

 userEntry.setDefaultScope(isDefaultScope);
 userEntry.setAccessControlType(AccessControlType.USER);
 userEntry.setEntityId("xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx");
 userEntry.setPermissions(userPermission);

 pathAccessControlEntries.add(userEntry);

 directoryClient.setAccessControlRecursive(pathAccessControlEntries);
}

```

## Update ACLs

When you *update* an ACL, you modify the ACL instead of replacing the ACL. For example, you can add a new security principal to the ACL without affecting other security principals listed in the ACL. To replace the ACL

instead of update it, see the [Set ACLs](#) section of this article.

This section shows you how to:

- Update an ACL
- Update ACLs recursively

## Update an ACL

First, get the ACL of a directory by calling the [PathAccessControl.getAccessControlList](#) method. Copy the list of ACL entries to a new [List](#) object of type [PathAccessControlEntry](#). Then locate the entry that you want to update and replace it in the list. Set the ACL by calling the [DataLakeDirectoryClient.setAccessControlList](#) method.

This example updates the ACL of a directory named `my-parent-directory` by replacing the entry for all other users.

```
public void UpdateACL(DataLakeFileSystemClient fileSystemClient, Boolean isDefaultScope){

 DataLakeDirectoryClient directoryClient =
 fileSystemClient.getDirectoryClient("my-parent-directory");

 List<PathAccessControlEntry> pathAccessControlEntries =
 directoryClient.getAccessControl().getAccessControlList();

 int index = -1;

 for (PathAccessControlEntry pathAccessControlEntry : pathAccessControlEntries){

 if (pathAccessControlEntry.getAccessControlType() == AccessControlType.OTHER){
 index = pathAccessControlEntries.indexOf(pathAccessControlEntry);
 break;
 }
 }

 if (index > -1){

 PathAccessControlEntry userEntry = new PathAccessControlEntry();

 RolePermissions userPermission = new RolePermissions();
 userPermission.setExecutePermission(true).setReadPermission(true).setWritePermission(true);

 userEntry.setDefaultScope(isDefaultScope);
 userEntry.setAccessControlType(AccessControlType.OTHER);
 userEntry.setPermissions(userPermission);

 pathAccessControlEntries.set(index, userEntry);
 }

 directoryClient.setAccessControlList(pathAccessControlEntries,
 directoryClient.getAccessControl().getGroup(),
 directoryClient.getAccessControl().getOwner());
}

}
```

You can also get and set the ACL of the root directory of a container. To get the root directory, pass an empty string (`""`) into the [DataLakeFileSystemClient.getDirectoryClient](#) method.

## Update ACLs recursively

To update an ACL recursively, create a new ACL object with the ACL entry that you want to update, and then use that object in update ACL operation. Do not get the existing ACL, just provide ACL entries to be updated.

Update ACLs recursively by calling the [DataLakeDirectoryClient.updateAccessControlRecursive](#) method. Pass this method a [List](#) of [PathAccessControlEntry](#) objects. Each [PathAccessControlEntry](#) defines an ACL entry.

If you want to update a **default** ACL entry, then you can the `setDefaultScope` method of the `PathAccessControlEntry` and pass in a value of `true`.

This example updates an ACL entry with write permission. This method accepts a boolean parameter named `isDefaultScope` that specifies whether to update the default ACL. That parameter is used in the call to the `setDefaultScope` method of the `PathAccessControlEntry`.

```
public void UpdateACLRecursively(DataLakeFileSystemClient fileSystemClient, Boolean isDefaultScope){

 DataLakeDirectoryClient directoryClient =
 fileSystemClient.getDirectoryClient("my-parent-directory");

 List<PathAccessControlEntry> pathAccessControlEntries =
 new ArrayList<PathAccessControlEntry>();

 // Create named user entry.
 PathAccessControlEntry userEntry = new PathAccessControlEntry();

 RolePermissions userPermission = new RolePermissions();
 userPermission.setExecutePermission(true).setReadPermission(true).setWritePermission(true);

 userEntry.setDefaultScope(isDefaultScope);
 userEntry.setAccessControlType(AccessControlType.USER);
 userEntry.setEntityId("xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx");
 userEntry.setPermissions(userPermission);

 pathAccessControlEntries.add(userEntry);

 directoryClient.updateAccessControlRecursive(pathAccessControlEntries);

}
```

## Remove ACL entries

You can remove one or more ACL entries. This section shows you how to:

- Remove an ACL entry
- Remove ACL entries recursively

### Remove an ACL entry

First, get the ACL of a directory by calling the `PathAccessControl.getAccessControlList` method. Copy the list of ACL entries to a new `List` object of type `PathAccessControlListEntry`. Then locate the entry that you want to remove and call the `Remove` method of the `List` object. Set the updated ACL by calling the `DataLakeDirectoryClient.setAccessControlList` method.

```

public void RemoveACLEntry(DataLakeFileSystemClient fileSystemClient, Boolean isDefaultScope){

 DataLakeDirectoryClient directoryClient =
 fileSystemClient.getDirectoryClient("my-parent-directory");

 List<PathAccessControlEntry> pathAccessControlEntries =
 directoryClient.getAccessControl().getAccessControlList();

 PathAccessControlEntry entryToRemove = null;

 for (PathAccessControlEntry pathAccessControlEntry : pathAccessControlEntries){

 if (pathAccessControlEntry.getEntityId() != null){

 if (pathAccessControlEntry.getEntityId().equals("xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx")){
 entryToRemove = pathAccessControlEntry;
 break;
 }
 }
 }

 if (entryToRemove != null){

 pathAccessControlEntries.remove(entryToRemove);

 directoryClient.setAccessControlList(pathAccessControlEntries,
 directoryClient.getAccessControl().getGroup(),
 directoryClient.getAccessControl().getOwner());
 }
}

```

## Remove ACL entries recursively

To remove ACL entries recursively, create a new ACL object for ACL entry to be removed, and then use that object in remove ACL operation. Do not get the existing ACL, just provide the ACL entries to be removed.

Remove ACL entries by calling the `DataLakeDirectoryClient.removeAccessControlRecursive` method. Pass this method a [List of `PathAccessControlEntry`](#) objects. Each `PathAccessControlEntry` defines an ACL entry.

If you want to remove a **default** ACL entry, then you can the `setDefaultScope` method of the `PathAccessControlEntry` and pass in a value of `true`.

This example removes an ACL entry from the ACL of the directory named `my-parent-directory`. This method accepts a boolean parameter named `isDefaultScope` that specifies whether to remove the entry from the default ACL. That parameter is used in the call to the `setDefaultScope` method of the [`PathAccessControlEntry`](#).

```
public void RemoveACLEntryRecursively(DataLakeFileSystemClient fileSystemClient, Boolean isDefaultScope){

 DataLakeDirectoryClient directoryClient =
 fileSystemClient.getDirectoryClient("my-parent-directory");

 List<PathRemoveAccessControlEntry> pathRemoveAccessControlEntries =
 new ArrayList<PathRemoveAccessControlEntry>();

 // Create named user entry.
 PathRemoveAccessControlEntry userEntry = new PathRemoveAccessControlEntry();

 RolePermissions userPermission = new RolePermissions();
 userPermission.setExecutePermission(true).setReadPermission(true).setWritePermission(true);

 userEntry.setDefaultScope(isDefaultScope);
 userEntry.setAccessControlType(AccessControlType.USER);
 userEntry.setEntityId("xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx");

 pathRemoveAccessControlEntries.add(userEntry);

 directoryClient.removeAccessControlRecursive(pathRemoveAccessControlEntries);

}
```

## Recover from failures

You might encounter runtime or permission errors. For runtime errors, restart the process from the beginning. Permission errors can occur if the security principal doesn't have sufficient permission to modify the ACL of a directory or file that is in the directory hierarchy being modified. Address the permission issue, and then choose to either resume the process from the point of failure by using a continuation token, or restart the process from beginning. You don't have to use the continuation token if you prefer to restart from the beginning. You can reapply ACL entries without any negative impact.

This example returns a continuation token in the event of a failure. The application can call this example method again after the error has been addressed, and pass in the continuation token. If this example method is called for the first time, the application can pass in a value of `null` for the continuation token parameter.

```

public String ResumeSetACLRecursively(DataLakeFileSystemClient fileSystemClient,
 DataLakeDirectoryClient directoryClient,
 List<PathAccessControlEntry> accessControlList,
 String continuationToken){

 try{
 PathSetAccessControlRecursiveOptions options = new
 PathSetAccessControlRecursiveOptions(accessControlList);

 options.setContinuationToken(continuationToken);

 Response<AccessControlChangeResult> accessControlChangeResult =
 directoryClient.setAccessControlRecursiveWithResponse(options, null, null);

 if (accessControlChangeResult.getValue().getCounters().getFailedChangesCount() > 0)
 {
 continuationToken =
 accessControlChangeResult.getValue().getContinuationToken();
 }

 return continuationToken;
 }
 catch(Exception ex){

 System.out.println(ex.toString());
 return continuationToken;
 }
}

```

If you want the process to complete uninterrupted by permission errors, you can specify that.

To ensure that the process completes uninterrupted, call the **setContinueOnFailure** method of a [PathSetAccessControlRecursiveOptions](#) object and pass in a value of **true**.

This example sets ACL entries recursively. If this code encounters a permission error, it records that failure and continues execution. This example prints the number of failures to the console.

```

public void ContinueOnFailure(DataLakeFileSystemClient fileSystemClient,
 DataLakeDirectoryClient directoryClient,
 List<PathAccessControlEntry> accessControlList){

 PathSetAccessControlRecursiveOptions options =
 new PathSetAccessControlRecursiveOptions(accessControlList);

 options.setContinueOnFailure(true);

 Response<AccessControlChangeResult> accessControlChangeResult =
 directoryClient.setAccessControlRecursiveWithResponse(options, null, null);

 AccessControlChangeCounters counters = accessControlChangeResult.getValue().getCounters();

 System.out.println("Number of directories changes: " +
 counters.getChangedDirectoriesCount());

 System.out.println("Number of files changed: " +
 counters.getChangedDirectoriesCount());

 System.out.println("Number of failures: " +
 counters.getChangedDirectoriesCount());
}

```

# Best practices

This section provides you some best practice guidelines for setting ACLs recursively.

## Handling runtime errors

A runtime error can occur for many reasons (For example: an outage or a client connectivity issue). If you encounter a runtime error, restart the recursive ACL process. ACLs can be reapplied to items without causing a negative impact.

## Handling permission errors (403)

If you encounter an access control exception while running a recursive ACL process, your AD [security principal](#) might not have sufficient permission to apply an ACL to one or more of the child items in the directory hierarchy. When a permission error occurs, the process stops and a continuation token is provided. Fix the permission issue, and then use the continuation token to process the remaining dataset. The directories and files that have already been successfully processed won't have to be processed again. You can also choose to restart the recursive ACL process. ACLs can be reapplied to items without causing a negative impact.

## Credentials

We recommend that you provision an Azure AD security principal that has been assigned the [Storage Blob Data Owner](#) role in the scope of the target storage account or container.

## Performance

To reduce latency, we recommend that you run the recursive ACL process in an Azure Virtual Machine (VM) that is located in the same region as your storage account.

## ACL limits

The maximum number of ACLs that you can apply to a directory or file is 32 access ACLs and 32 default ACLs. For more information, see [Access control in Azure Data Lake Storage Gen2](#).

## See also

- [API reference documentation](#)
- [Package \(Maven\)](#)
- [Samples](#)
- [Gen1 to Gen2 mapping](#)
- [Known issues](#)
- [Give Feedback](#)
- [Access control model in Azure Data Lake Storage Gen2](#)
- [Access control lists \(ACLs\) in Azure Data Lake Storage Gen2](#)

# Use Python to manage ACLs in Azure Data Lake Storage Gen2

8/22/2022 • 10 minutes to read • [Edit Online](#)

This article shows you how to use the Python to get, set, and update the access control lists of directories and files.

ACL inheritance is already available for new child items that are created under a parent directory. But you can also add, update, and remove ACLs recursively on the existing child items of a parent directory without having to make these changes individually for each child item.

[Package \(Python Package Index\)](#) | [Samples](#) | [Recursive ACL samples](#) | [API reference](#) | [Gen1 to Gen2 mapping](#) | [Give Feedback](#)

## Prerequisites

- An Azure subscription. For more information, see [Get Azure free trial](#).
- A storage account that has hierarchical namespace (HNS) enabled. Follow [these](#) instructions to create one.
- Azure CLI version `2.6.0` or higher.
- One of the following security permissions:
  - A provisioned Azure Active Directory (AD) [security principal](#) that has been assigned the [Storage Blob Data Owner](#) role in the scope of the either the target container, parent resource group or subscription.
  - Owning user of the target container or directory to which you plan to apply ACL settings. To set ACLs recursively, this includes all child items in the target container or directory.
  - Storage account key.

## Set up your project

Install the Azure Data Lake Storage client library for Python by using [pip](#).

```
pip install azure-storage-file-datalake
```

Add these import statements to the top of your code file.

```
import os, uuid, sys
from azure.storage.filedatalake import DataLakeServiceClient
from azure.core._match_conditions import MatchConditions
from azure.storage.filedatalake._models import ContentSettings
```

## Connect to the account

To use the snippets in this article, you'll need to create a `DataLakeServiceClient` instance that represents the storage account.

## Connect by using Azure Active Directory (AD)

### NOTE

If you're using Azure Active Directory (Azure AD) to authorize access, then make sure that your security principal has been assigned the [Storage Blob Data Owner role](#). To learn more about how ACL permissions are applied and the effects of changing them, see [Access control model in Azure Data Lake Storage Gen2](#).

You can use the [Azure identity client library for Python](#) to authenticate your application with Azure AD.

Get a client ID, a client secret, and a tenant ID. To do this, see [Acquire a token from Azure AD for authorizing requests from a client application](#). As part of that process, you'll have to assign one of the following [Azure role-based access control \(Azure RBAC\)](#) roles to your security principal.

ROLE	ACL SETTING CAPABILITY
Storage Blob Data Owner	All directories and files in the account.
Storage Blob Data Contributor	Only directories and files owned by the security principal.

This example creates a `DataLakeServiceClient` instance by using a client ID, a client secret, and a tenant ID.

```
def initialize_storage_account_ad(storage_account_name, client_id, client_secret, tenant_id):

 try:
 global service_client

 credential = ClientSecretCredential(tenant_id, client_id, client_secret)

 service_client = DataLakeServiceClient(account_url="{}://{}.dfs.core.windows.net".format(
 "https", storage_account_name), credential=credential)

 except Exception as e:
 print(e)
```

### NOTE

For more examples, see the [Azure identity client library for Python](#) documentation.

## Connect by using an account key

This is the easiest way to connect to an account.

This example creates a `DataLakeServiceClient` instance by using an account key.

```
def initialize_storage_account(storage_account_name, storage_account_key):

 try:
 global service_client

 service_client = DataLakeServiceClient(account_url="{}://{}.dfs.core.windows.net".format(
 "https", storage_account_name), credential=storage_account_key)

 except Exception as e:
 print(e)
```

- Replace the `storage_account_name` placeholder value with the name of your storage account.

- Replace the `storage_account_key` placeholder value with your storage account access key.

## Set ACLs

When you *set* an ACL, you *replace* the entire ACL including all of its entries. If you want to change the permission level of a security principal or add a new security principal to the ACL without affecting other existing entries, you should *update* the ACL instead. To update an ACL instead of replace it, see the [Update ACLs](#) section of this article.

This section shows you how to:

- Set the ACL of a directory
- Set the ACL of a file

### Set the ACL of a directory

Get the access control list (ACL) of a directory by calling the `DataLakeDirectoryClient.get_access_control` method and set the ACL by calling the `DataLakeDirectoryClient.set_access_control` method.

This example gets and sets the ACL of a directory named `my-directory`. The string `rwxr-xrw-` gives the owning user read, write, and execute permissions, gives the owning group only read and execute permissions, and gives all others read and write permission.

```
def manage_directory_permissions():
 try:
 file_system_client = service_client.get_file_system_client(file_system="my-file-system")

 directory_client = file_system_client.get_directory_client("my-directory")

 acl_props = directory_client.get_access_control()

 print(acl_props['permissions'])

 new_dir_permissions = "rwxr-xrw-"

 directory_client.set_access_control(permissions=new_dir_permissions)

 acl_props = directory_client.get_access_control()

 print(acl_props['permissions'])

 except Exception as e:
 print(e)
```

You can also get and set the ACL of the root directory of a container. To get the root directory, call the `FileSystemClient._get_root_directory_client` method.

### Set the ACL of a file

Get the access control list (ACL) of a file by calling the `DataLakeFileClient.get_access_control` method and set the ACL by calling the `DataLakeFileClient.set_access_control` method.

This example gets and sets the ACL of a file named `my-file.txt`. The string `rwxr-xrw-` gives the owning user read, write, and execute permissions, gives the owning group only read and execute permissions, and gives all others read and write permission.

```

def manage_file_permissions():
 try:
 file_system_client = service_client.get_file_system_client(file_system="my-file-system")

 directory_client = file_system_client.get_directory_client("my-directory")

 file_client = directory_client.get_file_client("uploaded-file.txt")

 acl_props = file_client.get_access_control()

 print(acl_props['permissions'])

 new_file_permissions = "rwxr-xrw-"

 file_client.set_access_control(permissions=new_file_permissions)

 acl_props = file_client.get_access_control()

 print(acl_props['permissions'])

 except Exception as e:
 print(e)

```

## Set ACLs recursively

When you *set* an ACL, you **replace** the entire ACL including all of its entries. If you want to change the permission level of a security principal or add a new security principal to the ACL without affecting other existing entries, you should *update* the ACL instead. To update an ACL instead of replace it, see the [Update ACLs recursively](#) section of this article.

Set ACLs recursively by calling the `DataLakeDirectoryClient.set_access_control_recursive` method.

If you want to set a **default** ACL entry, then add the string `default:` to the beginning of each ACL entry string.

This example sets the ACL of a directory named `my-parent-directory`.

This method accepts a boolean parameter named `is_default_scope` that specifies whether to set the default ACL. If that parameter is `True`, the list of ACL entries are preceded with the string `default:`.

The entries of the ACL give the owning user read, write, and execute permissions, gives the owning group only read and execute permissions, and gives all others no access. The last ACL entry in this example gives a specific user with the object ID `"xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"` read and execute permissions. These entries give the owning user read, write, and execute permissions, gives the owning group only read and execute permissions, and gives all others no access. The last ACL entry in this example gives a specific user with the object ID `"xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx"` read and execute permissions.

```
def set_permission_recursively(is_default_scope):

 try:
 file_system_client = service_client.get_file_system_client(file_system="my-container")

 directory_client = file_system_client.get_directory_client("my-parent-directory")

 acl = 'user::rwx,group::rwx,other::rwx,user:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx:r--'

 if is_default_scope:
 acl = 'default:user::rwx,default:group::rwx,default:other::rwx,default:user:xxxxxxxx-xxxx-xxxx-
xxxx-xxxxxxxxxxx:r--'

 directory_client.set_access_control_recursive(acl=acl)

 acl_props = directory_client.get_access_control()

 print(acl_props['permissions'])

 except Exception as e:
 print(e)
```

To see an example that processes ACLs recursively in batches by specifying a batch size, see the Python [sample](#).

## Update ACLs recursively

When you *update* an ACL, you modify the ACL instead of replacing the ACL. For example, you can add a new security principal to the ACL without affecting other security principals listed in the ACL. To replace the ACL instead of update it, see the [Set ACLs](#) section of this article.

To update an ACL recursively, create a new ACL object with the ACL entry that you want to update, and then use that object in update ACL operation. Do not get the existing ACL, just provide ACL entries to be updated. Update an ACL recursively by calling the `DataLakeDirectoryClient.update_access_control_recursive` method. If you want to update a `default` ACL entry, then add the string `default:` to the beginning of each ACL entry string.

This example updates an ACL entry with write permission.

This example sets the ACL of a directory named `my-parent-directory`. This method accepts a boolean parameter named `is_default_scope` that specifies whether to update the default ACL. If that parameter is `True`, the updated ACL entry is preceded with the string `default:`.

```

def update_permission_recursively(is_default_scope):

 try:
 file_system_client = service_client.get_file_system_client(file_system="my-container")

 directory_client = file_system_client.get_directory_client("my-parent-directory")

 acl = 'user:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx:rwx'

 if is_default_scope:
 acl = 'default:user:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx:rwx'

 directory_client.update_access_control_recursive(acl=acl)

 acl_props = directory_client.get_access_control()

 print(acl_props['permissions'])

 except Exception as e:
 print(e)

```

To see an example that processes ACLs recursively in batches by specifying a batch size, see the Python [sample](#).

## Remove ACL entries recursively

You can remove one or more ACL entries. To remove ACL entries recursively, create a new ACL object for ACL entry to be removed, and then use that object in remove ACL operation. Do not get the existing ACL, just provide the ACL entries to be removed.

Remove ACL entries by calling the `DataLakeDirectoryClient.remove_access_control_recursive` method. If you want to remove a `default` ACL entry, then add the string `default:` to the beginning of the ACL entry string.

This example removes an ACL entry from the ACL of the directory named `my-parent-directory`. This method accepts a boolean parameter named `is_default_scope` that specifies whether to remove the entry from the default ACL. If that parameter is `True`, the updated ACL entry is preceded with the string `default:`.

```

def remove_permission_recursively(is_default_scope):

 try:
 file_system_client = service_client.get_file_system_client(file_system="my-container")

 directory_client = file_system_client.get_directory_client("my-parent-directory")

 acl = 'user:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'

 if is_default_scope:
 acl = 'default:user:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx'

 directory_client.remove_access_control_recursive(acl=acl)

 except Exception as e:
 print(e)

```

To see an example that processes ACLs recursively in batches by specifying a batch size, see the Python [sample](#).

## Recover from failures

You might encounter runtime or permission errors. For runtime errors, restart the process from the beginning. Permission errors can occur if the security principal doesn't have sufficient permission to modify the ACL of a directory or file that is in the directory hierarchy being modified. Address the permission issue, and then choose

to either resume the process from the point of failure by using a continuation token, or restart the process from beginning. You don't have to use the continuation token if you prefer to restart from the beginning. You can reapply ACL entries without any negative impact.

This example returns a continuation token in the event of a failure. The application can call this example method again after the error has been addressed, and pass in the continuation token. If this example method is called for the first time, the application can pass in a value of `None` for the continuation token parameter.

```
def resume_set_acl_recursive(continuation_token):

 try:
 file_system_client = service_client.get_file_system_client(file_system="my-container")

 directory_client = file_system_client.get_directory_client("my-parent-directory")

 acl = 'user::rwx,group::rwx,other::rwx,user:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx:r--'

 acl_change_result = directory_client.set_access_control_recursive(acl=acl,
continuation=continuation_token)

 continuation_token = acl_change_result.continuation

 return continuation_token

except Exception as e:
 print(e)
 return continuation_token
```

To see an example that processes ACLs recursively in batches by specifying a batch size, see the Python [sample](#).

If you want the process to complete uninterrupted by permission errors, you can specify that.

To ensure that the process completes uninterrupted, don't pass a continuation token into the `DataLakeDirectoryClient.set_access_control_recursive` method.

This example sets ACL entries recursively. If this code encounters a permission error, it records that failure and continues execution. This example prints the number of failures to the console.

```
def continue_on_failure():

 try:
 file_system_client = service_client.get_file_system_client(file_system="my-container")

 directory_client = file_system_client.get_directory_client("my-parent-directory")

 acl = 'user::rwx,group::rwx,other::rwx,user:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx:r--'

 acl_change_result = directory_client.set_access_control_recursive(acl=acl)

 print("Summary: {} directories and {} files were updated successfully, {} failures were counted."
 .format(acl_change_result.counters.directories_successful,
acl_change_result.counters.files_successful,
 acl_change_result.counters.failure_count))

 except Exception as e:
 print(e)
```

To see an example that processes ACLs recursively in batches by specifying a batch size, see the Python [sample](#).

## Best practices

This section provides you some best practice guidelines for setting ACLs recursively.

#### **Handling runtime errors**

A runtime error can occur for many reasons (For example: an outage or a client connectivity issue). If you encounter a runtime error, restart the recursive ACL process. ACLs can be reapplied to items without causing a negative impact.

#### **Handling permission errors (403)**

If you encounter an access control exception while running a recursive ACL process, your AD [security principal](#) might not have sufficient permission to apply an ACL to one or more of the child items in the directory hierarchy. When a permission error occurs, the process stops and a continuation token is provided. Fix the permission issue, and then use the continuation token to process the remaining dataset. The directories and files that have already been successfully processed won't have to be processed again. You can also choose to restart the recursive ACL process. ACLs can be reapplied to items without causing a negative impact.

#### **Credentials**

We recommend that you provision an Azure AD security principal that has been assigned the [Storage Blob Data Owner](#) role in the scope of the target storage account or container.

#### **Performance**

To reduce latency, we recommend that you run the recursive ACL process in an Azure Virtual Machine (VM) that is located in the same region as your storage account.

#### **ACL limits**

The maximum number of ACLs that you can apply to a directory or file is 32 access ACLs and 32 default ACLs. For more information, see [Access control in Azure Data Lake Storage Gen2](#).

## See also

- [API reference documentation](#)
- [Package \(Python Package Index\)](#)
- [Samples](#)
- [Gen1 to Gen2 mapping](#)
- [Known issues](#)
- [Give Feedback](#)
- [Access control model in Azure Data Lake Storage Gen2](#)
- [Access control lists \(ACLs\) in Azure Data Lake Storage Gen2](#)

# Use JavaScript SDK in Node.js to manage ACLs in Azure Data Lake Storage Gen2

8/22/2022 • 4 minutes to read • [Edit Online](#)

This article shows you how to use Node.js to get, set, and update the access control lists of directories and files.

[Package \(Node Package Manager\)](#) | [Samples](#) | [Give Feedback](#)

## Prerequisites

- An Azure subscription. For more information, see [Get Azure free trial](#).
- A storage account that has hierarchical namespace (HNS) enabled. Follow [these](#) instructions to create one.
- Azure CLI version `2.6.0` or higher.
- One of the following security permissions:
  - A provisioned Azure Active Directory (AD) [security principal](#) that has been assigned the [Storage Blob Data Owner](#) role in the scope of the either the target container, parent resource group or subscription.
  - Owning user of the target container or directory to which you plan to apply ACL settings. To set ACLs recursively, this includes all child items in the target container or directory.
  - Storage account key..

## Set up your project

Install Data Lake client library for JavaScript by opening a terminal window, and then typing the following command.

```
npm install @azure/storage-file-datalake
```

Import the `storage-file-datalake` package by placing this statement at the top of your code file.

```
const {
 AzureStorageDataLake,
 DataLakeServiceClient,
 StorageSharedKeyCredential
} = require("@azure/storage-file-datalake");
```

## Connect to the account

To use the snippets in this article, you'll need to create a `DataLakeServiceClient` instance that represents the storage account.

### Connect by using Azure Active Directory (AD)

#### NOTE

If you're using Azure Active Directory (Azure AD) to authorize access, then make sure that your security principal has been assigned the [Storage Blob Data Owner role](#). To learn more about how ACL permissions are applied and the effects of changing them, see [Access control model in Azure Data Lake Storage Gen2](#).

You can use the [Azure identity client library for JS](#) to authenticate your application with Azure AD.

Get a client ID, a client secret, and a tenant ID. To do this, see [Acquire a token from Azure AD for authorizing requests from a client application](#). As part of that process, you'll have to assign one of the following [Azure role-based access control \(Azure RBAC\)](#) roles to your security principal.

ROLE	ACL SETTING CAPABILITY
Storage Blob Data Owner	All directories and files in the account.
Storage Blob Data Contributor	Only directories and files owned by the security principal.

This example creates a **DataLakeServiceClient** instance by using a client ID, a client secret, and a tenant ID.

```
function GetDataLakeServiceClientAD(accountName, clientID, clientSecret, tenantID) {

 const credential = new ClientSecretCredential(tenantID, clientID, clientSecret);

 const datalakeServiceClient = new DataLakeServiceClient(
 `https://${accountName}.dfs.core.windows.net`, credential);

 return datalakeServiceClient;
}
```

#### NOTE

For more examples, see the [Azure identity client library for JS](#) documentation.

### Connect by using an account key

This is the easiest way to connect to an account.

This example creates a **DataLakeServiceClient** instance by using an account key.

```
function GetDataLakeServiceClient(accountName, accountKey) {

 const sharedKeyCredential =
 new StorageSharedKeyCredential(accountName, accountKey);

 const datalakeServiceClient = new DataLakeServiceClient(
 `https://${accountName}.dfs.core.windows.net`, sharedKeyCredential);

 return datalakeServiceClient;
}
```

**NOTE**

This method of authorization works only for Node.js applications. If you plan to run your code in a browser, you can authorize by using Azure Active Directory (AD).

## Get and set a directory ACL

This example gets and then sets the ACL of a directory named `my-directory`. This example gives the owning user read, write, and execute permissions, gives the owning group only read and execute permissions, and gives all others read access.

**NOTE**

If your application authorizes access by using Azure Active Directory (Azure AD), then make sure that the security principal that your application uses to authorize access has been assigned the [Storage Blob Data Owner role](#). To learn more about how ACL permissions are applied and the effects of changing them, see [Access control in Azure Data Lake Storage Gen2](#).

```

async function ManageDirectoryACls(fileSystemClient) {

 const directoryClient = fileSystemClient.getDirectoryClient("my-directory");
 const permissions = await directoryClient.getAccessControl();

 console.log(permissions.acl);

 const acl = [
 {
 accessControlType: "user",
 entityId: "",
 defaultScope: false,
 permissions: {
 read: true,
 write: true,
 execute: true
 }
 },
 {
 accessControlType: "group",
 entityId: "",
 defaultScope: false,
 permissions: {
 read: true,
 write: false,
 execute: true
 }
 },
 {
 accessControlType: "other",
 entityId: "",
 defaultScope: false,
 permissions: {
 read: true,
 write: true,
 execute: false
 }
 }
]

 await directoryClient.setAccessControl(acl);
}

```

You can also get and set the ACL of the root directory of a container. To get the root directory, pass an empty string ( / ) into the `DataLakeFileSystemClient.getDirectoryClient` method.

## Get and set a file ACL

This example gets and then sets the ACL of a file named `upload-file.txt`. This example gives the owning user read, write, and execute permissions, gives the owning group only read and execute permissions, and gives all others read access.

### NOTE

If your application authorizes access by using Azure Active Directory (Azure AD), then make sure that the security principal that your application uses to authorize access has been assigned the [Storage Blob Data Owner](#) role. To learn more about how ACL permissions are applied and the effects of changing them, see [Access control in Azure Data Lake Storage Gen2](#).

```
async function ManageFileACls(fileSystemClient) {

 const fileClient = fileSystemClient.getFileClient("my-directory/uploaded-file.txt");
 const permissions = await fileClient.getAccessControl();

 console.log(permissions.acl);

 const acl = [
 {
 accessControlType: "user",
 entityId: "",
 defaultScope: false,
 permissions: {
 read: true,
 write: true,
 execute: true
 }
 },
 {
 accessControlType: "group",
 entityId: "",
 defaultScope: false,
 permissions: {
 read: true,
 write: false,
 execute: true
 }
 },
 {
 accessControlType: "other",
 entityId: "",
 defaultScope: false,
 permissions: {
 read: true,
 write: true,
 execute: false
 }
 }
];

 await fileClient.setAccessControl(acl);
}
```

## See also

- [Package \(Node Package Manager\)](#)
- [Samples](#)
- [Give Feedback](#)
- [Access control model in Azure Data Lake Storage Gen2](#)
- [Access control lists \(ACLs\) in Azure Data Lake Storage Gen2](#)

# Use Azure Storage Explorer to manage directories and files in Azure Data Lake Storage Gen2

8/22/2022 • 3 minutes to read • [Edit Online](#)

This article shows you how to use [Azure Storage Explorer](#) to create and manage directories and files in storage accounts that has hierarchical namespace (HNS) enabled.

## Prerequisites

- An Azure subscription. See [Get Azure free trial](#).
- A storage account that has hierarchical namespace (HNS) enabled. Follow [these](#) instructions to create one.
- Azure Storage Explorer installed on your local computer. To install Azure Storage Explorer for Windows, Macintosh, or Linux, see [Azure Storage Explorer](#).

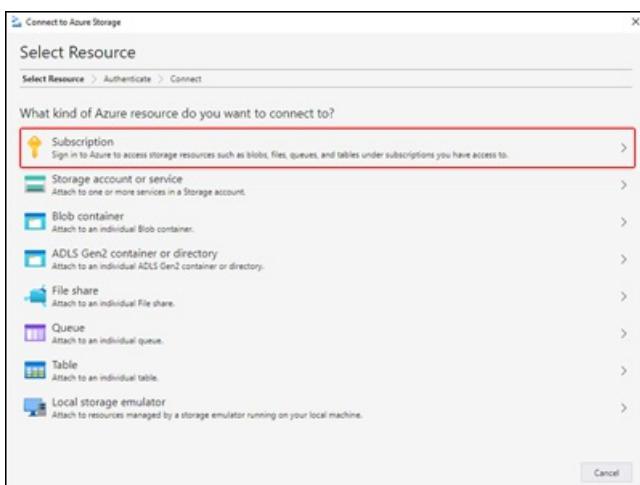
### NOTE

Storage Explorer makes use of both the Blob (blob) & Data Lake Storage Gen2 (dfs) [endpoints](#) when working with Azure Data Lake Storage Gen2. If access to Azure Data Lake Storage Gen2 is configured using private endpoints, ensure that two private endpoints are created for the storage account: one with the target sub-resource `blob` and the other with the target sub-resource `dfs`.

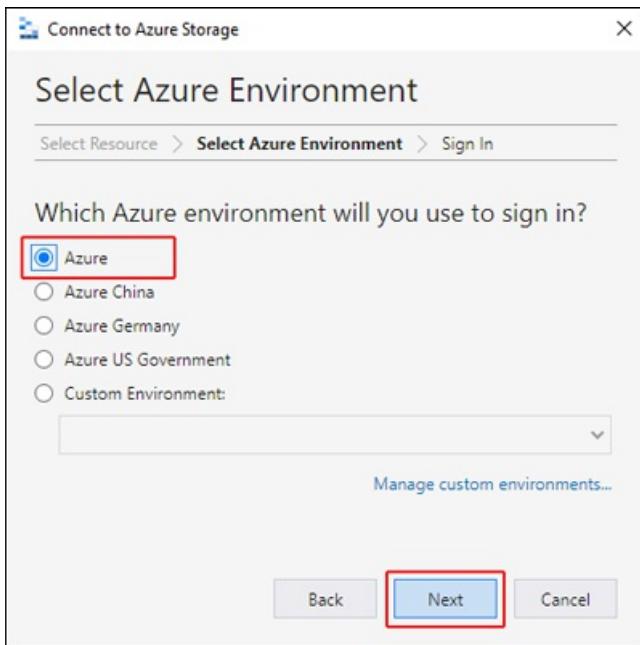
## Sign in to Storage Explorer

When you first start Storage Explorer, the **Microsoft Azure Storage Explorer - Connect to Azure Storage** window appears. While Storage Explorer provides several ways to connect to storage accounts, only one way is currently supported for managing ACLs.

In the **Select Resource** panel, select **Subscription**.

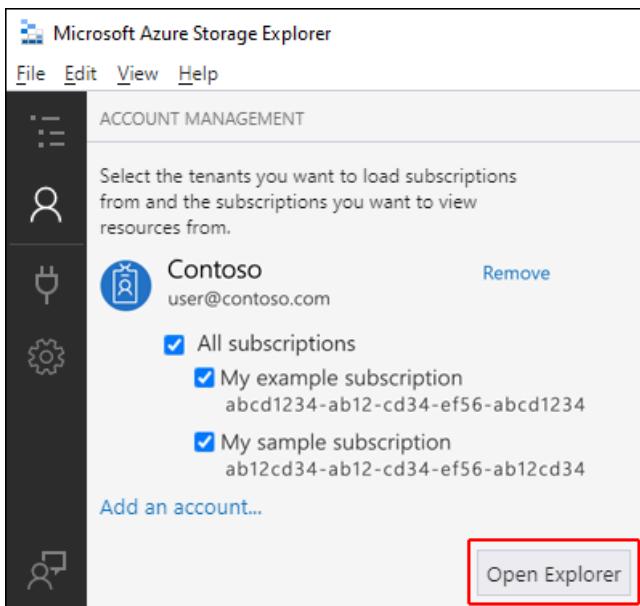


In the **Select Azure Environment** panel, select an Azure environment to sign in to. You can sign in to global Azure, a national cloud or an Azure Stack instance. Then select **Next**.

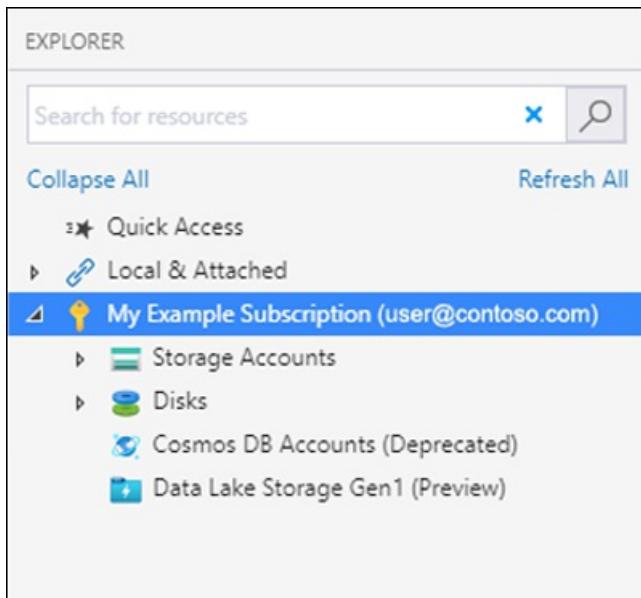


Storage Explorer will open a webpage for you to sign in.

After you successfully sign in with an Azure account, the account and the Azure subscriptions associated with that account appear under **ACCOUNT MANAGEMENT**. Select the Azure subscriptions that you want to work with, and then select **Open Explorer**.

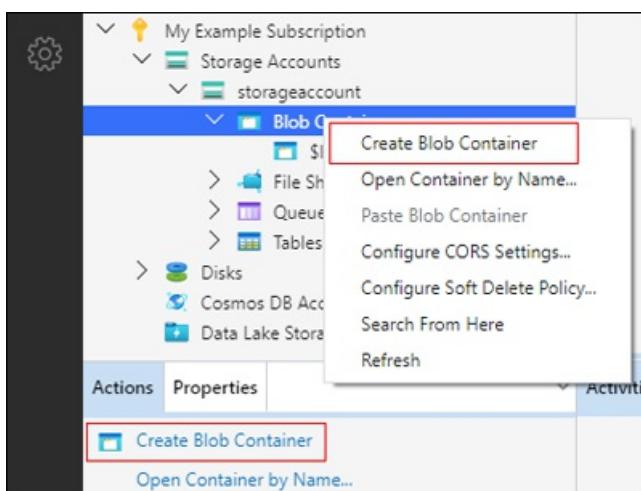


When it completes connecting, Azure Storage Explorer loads with the **Explorer** tab shown. This view gives you insight to all of your Azure storage accounts as well as local storage configured through the [Azurite storage emulator](#) or [Azure Stack](#) environments.

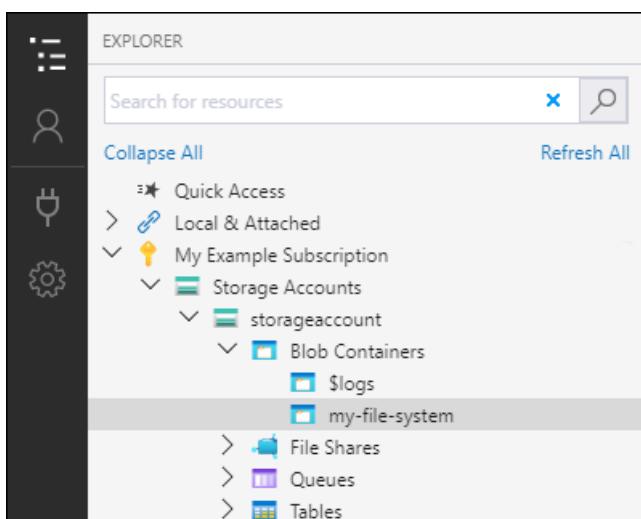


## Create a container

A container holds directories and files. To create one, expand the storage account you created in the proceeding step. Select **Blob Containers**, right-click, and select **Create Blob Container**. Alternatively, you can select **Blob Containers**, then select **Create Blob Container** in the Actions pane.

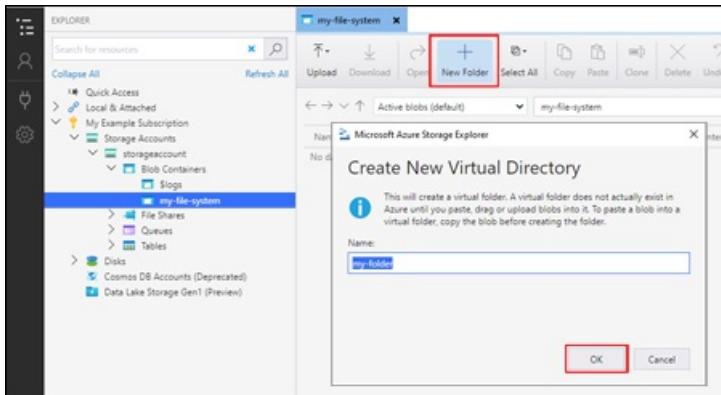


Enter the name for your container. See the [Create a container](#) section for a list of rules and restrictions on naming containers. When complete, press **Enter** to create the container. After the container has been successfully created, it is displayed under the **Blob Containers** folder for the selected storage account.



## Create a directory

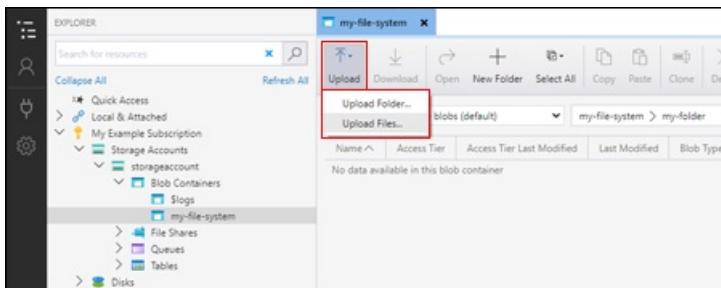
To create a directory, select the container that you created in the proceeding step. In the container ribbon, choose the **New Folder** button. Enter the name for your directory. When complete, press **Enter** to create the directory. After the directory has been successfully created, it appears in the editor window.



## Upload blobs to the directory

On the directory ribbon, choose the **Upload** button. This operation gives you the option to upload a folder or a file.

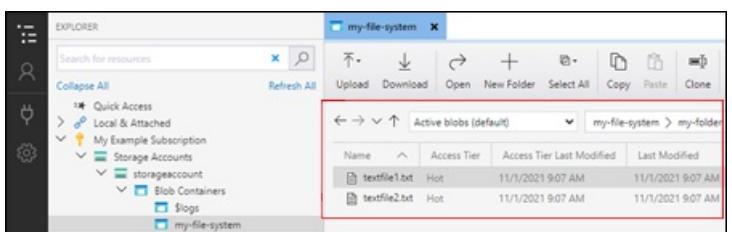
Choose the files or folder to upload.



When you select **Upload**, the files selected are queued, and each file is uploaded. When the upload is complete, the results are shown in the **Activities** window.

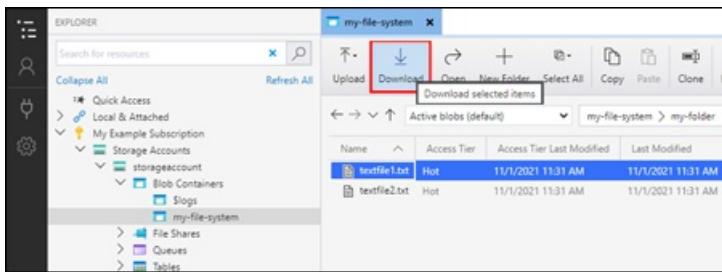
## View blobs in a directory

In the **Azure Storage Explorer** application, select a directory under a storage account. The main pane shows a list of the blobs in the selected directory.



## Download blobs

To download files by using **Azure Storage Explorer**, with a file selected, select **Download** from the ribbon. A file dialog opens and provides you the ability to enter a file name. Select **Select Folder** to start the download of a file to the local location.



## Next steps

Learn how to manage file and directory permission by setting access control lists (ACLs)

[Use Azure Storage Explorer to manage ACLs in Azure Data Lake Storage Gen2](#)

# Use PowerShell to manage directories and files in Azure Data Lake Storage Gen2

8/22/2022 • 5 minutes to read • [Edit Online](#)

This article shows you how to use PowerShell to create and manage directories and files in storage accounts that have a hierarchical namespace.

To learn about how to get, set, and update the access control lists (ACL) of directories and files, see [Use PowerShell to manage ACLs in Azure Data Lake Storage Gen2](#).

[Reference](#) | [Gen1 to Gen2 mapping](#) | [Give feedback](#)

## Prerequisites

- An Azure subscription. For more information, see [Get Azure free trial](#).
- A storage account that has hierarchical namespace enabled. Follow [these](#) instructions to create one.
- .NET Framework is 4.7.2 or greater installed. For more information, see [Download .NET Framework](#).
- PowerShell version [5.1](#) or higher.

## Install the PowerShell module

1. Verify that the version of PowerShell that have installed is [5.1](#) or higher by using the following command.

```
echo $PSVersionTable.PSVersion.ToString()
```

To upgrade your version of PowerShell, see [Upgrading existing Windows PowerShell](#)

2. Install **Az.Storage** module.

```
Install-Module Az.Storage -Repository PSGallery -Force
```

For more information about how to install PowerShell modules, see [Install the Azure PowerShell module](#)

## Connect to the account

Choose how you want your commands to obtain authorization to the storage account.

### Option 1: Obtain authorization by using Azure Active Directory (Azure AD)

With this approach, the system ensures that your user account has the appropriate Azure role-based access control (Azure RBAC) assignments and ACL permissions.

1. Open a Windows PowerShell command window, and then sign in to your Azure subscription with the [Connect-AzAccount](#) command and follow the on-screen directions.

```
Connect-AzAccount
```

2. If your identity is associated with more than one subscription, then set your active subscription to subscription of the storage account that you want create and manage directories in. In this example, replace the <subscription-id> placeholder value with the ID of your subscription.

```
Select-AzSubscription -SubscriptionId <subscription-id>
```

3. Get the storage account context.

```
$ctx = New-AzStorageContext -StorageAccountName '<storage-account-name>' -UseConnectedAccount
```

### Option 2: Obtain authorization by using the storage account key

With this approach, the system doesn't check Azure RBAC or ACL permissions. Get the storage account context by using an account key.

```
$ctx = New-AzStorageContext -StorageAccountName '<storage-account-name>' -StorageAccountKey '<storage-account-key>'
```

## Create a container

A container acts as a file system for your files. You can create one by using the `New-AzStorageContainer` cmdlet.

This example creates a container named `my-file-system`.

```
$filesystemName = "my-file-system"
New-AzStorageContainer -Context $ctx -Name $filesystemName
```

## Create a directory

Create a directory reference by using the `New-AzDataLakeGen2Item` cmdlet.

This example adds a directory named `my-directory` to a container.

```
$filesystemName = "my-file-system"
$dirname = "my-directory/"
New-AzDataLakeGen2Item -Context $ctx -FileSystem $filesystemName -Path $dirname -Directory
```

This example adds the same directory, but also sets the permissions, umask, property values, and metadata values.

```
$dir = New-AzDataLakeGen2Item -Context $ctx -FileSystem $filesystemName -Path $dirname -Directory -
Permission rwxrwxrwx -Umask ---rwx--- -Property @{"ContentEncoding" = "UTF8"; "CacheControl" = "READ"} -
Metadata @{"tag1" = "value1"; "tag2" = "value2" }
```

## Show directory properties

This example gets a directory by using the `Get-AzDataLakeGen2Item` cmdlet, and then prints property values to the console.

```
$filesystemName = "my-file-system"
$dirname = "my-directory/"
$dir = Get-AzDataLakeGen2Item -Context $ctx -FileSystem $filesystemName -Path $dirname
$dir.ACL
$dir.Permissions
$dir.Group
$dir.Owner
$dir.Properties
$dir.Properties.Metadata
```

#### NOTE

To get the root directory of the container, omit the `-Path` parameter.

## Rename or move a directory

Rename or move a directory by using the `Move-AzDataLakeGen2Item` cmdlet.

This example renames a directory from the name `my-directory` to the name `my-new-directory`.

```
$filesystemName = "my-file-system"
$dirname = "my-directory/"
$dirname2 = "my-new-directory/"
Move-AzDataLakeGen2Item -Context $ctx -FileSystem $filesystemName -Path $dirname -DestFileSystem
$filesystemName -DestPath $dirname2
```

#### NOTE

Use the `-Force` parameter if you want to overwrite without prompts.

This example moves a directory named `my-directory` to a subdirectory of `my-directory-2` named `my-subdirectory`.

```
$filesystemName = "my-file-system"
$dirname = "my-directory/"
$dirname2 = "my-directory-2/my-subdirectory/"
Move-AzDataLakeGen2Item -Context $ctx -FileSystem $filesystemName -Path $dirname -DestFileSystem
$filesystemName -DestPath $dirname2
```

## Delete a directory

Delete a directory by using the `Remove-AzDataLakeGen2Item` cmdlet.

This example deletes a directory named `my-directory`.

```
$filesystemName = "my-file-system"
$dirname = "my-directory/"
Remove-AzDataLakeGen2Item -Context $ctx -FileSystem $filesystemName -Path $dirname
```

You can use the `-Force` parameter to remove the file without a prompt.

## Download from a directory

Download a file from a directory by using the `Get-AzDataLakeGen2ItemContent` cmdlet.

This example downloads a file named `upload.txt` from a directory named `my-directory`.

```
$filesystemName = "my-file-system"
$filePath = "my-directory/upload.txt"
$downloadFilePath = "download.txt"
Get-AzDataLakeGen2ItemContent -Context $ctx -FileSystem $filesystemName -Path $filePath -Destination
$downloadFilePath
```

## List directory contents

List the contents of a directory by using the `Get-AzDataLakeGen2ChildItem` cmdlet. You can use the optional parameter `-OutputUserPrincipalName` to get the name (instead of the object ID) of users.

This example lists the contents of a directory named `my-directory`.

```
$filesystemName = "my-file-system"
$dirname = "my-directory/"
Get-AzDataLakeGen2ChildItem -Context $ctx -FileSystem $filesystemName -Path $dirname -
OutputUserPrincipalName
```

The following example lists the `ACL`, `Permissions`, `Group`, and `Owner` properties of each item in the directory. The `-FetchProperty` parameter is required to get values for the `ACL` property.

```
$filesystemName = "my-file-system"
$dirname = "my-directory/"
$properties = Get-AzDataLakeGen2ChildItem -Context $ctx -FileSystem $filesystemName -Path $dirname -Recurse
-FetchProperty
$properties.ACL
$properties.Permissions
$properties.Group
$properties.Owner
```

### NOTE

To list the contents of the root directory of the container, omit the `-Path` parameter.

## Upload a file to a directory

Upload a file to a directory by using the `New-AzDataLakeGen2Item` cmdlet.

This example uploads a file named `upload.txt` to a directory named `my-directory`.

```
$localSrcFile = "upload.txt"
$filesystemName = "my-file-system"
$dirname = "my-directory/"
$destPath = $dirname + (Get-Item $localSrcFile).Name
New-AzDataLakeGen2Item -Context $ctx -FileSystem $filesystemName -Path $destPath -Source $localSrcFile -
Force
```

This example uploads the same file, but then sets the permissions, umask, property values, and metadata values of the destination file. This example also prints these values to the console.

```

$file = New-AzDataLakeGen2Item -Context $ctx -FileSystem $filesystemName -Path $destPath -Source
$localSrcFile -Permission rwxrwxrwx -Umask ---rwx--- -Property @{"ContentEncoding" = "UTF8"; "CacheControl"
= "READ"} -Metadata @{"tag1" = "value1"; "tag2" = "value2" }
$file1
$file1.Properties
$file1.Properties.Metadata

```

#### NOTE

To upload a file to the root directory of the container, omit the `-Path` parameter.

## Show file properties

This example gets a file by using the `Get-AzDataLakeGen2Item` cmdlet, and then prints property values to the console.

```

$filepath = "my-directory/upload.txt"
$filesystemName = "my-file-system"
$file = Get-AzDataLakeGen2Item -Context $ctx -FileSystem $filesystemName -Path $filepath
$file
$file.ACL
$file.Permissions
$file.Group
$file.Owner
$file.Properties
$file.Properties.Metadata

```

## Delete a file

Delete a file by using the `Remove-AzDataLakeGen2Item` cmdlet.

This example deletes a file named `upload.txt`.

```

$filesystemName = "my-file-system"
$filepath = "upload.txt"
Remove-AzDataLakeGen2Item -Context $ctx -FileSystem $filesystemName -Path $filepath

```

You can use the `-Force` parameter to remove the file without a prompt.

## Gen1 to Gen2 Mapping

The following table shows how the cmdlets used for Data Lake Storage Gen1 map to the cmdlets for Data Lake Storage Gen2.

DATA LAKE STORAGE GEN1 CMDLET	DATA LAKE STORAGE GEN2 CMDLET	NOTES
Get-AzDataLakeStoreChildItem	Get-AzDataLakeGen2ChildItem	By default, the <code>Get-AzDataLakeGen2ChildItem</code> cmdlet only lists the first level child items. The <code>-Recurse</code> parameter lists child items recursively.

DATA LAKE STORAGE GEN1 CMDLET	DATA LAKE STORAGE GEN2 CMDLET	NOTES
Get-AzDataLakeStoreItem Get-AzDataLakeStoreItemAclEntry Get-AzDataLakeStoreItemOwner Get-AzDataLakeStoreItemPermission	Get-AzDataLakeGen2Item	The output items of the Get-AzDataLakeGen2Item cmdlet has these properties: Acl, Owner, Group, Permission.
Get-AzDataLakeStoreItemContent	Get-AzDataLakeGen2FileContent	The Get-AzDataLakeGen2FileContent cmdlet download file content to local file.
Move-AzDataLakeStoreItem	Move-AzDataLakeGen2Item	
New-AzDataLakeStoreItem	New-AzDataLakeGen2Item	This cmdlet uploads the new file content from a local file.
Remove-AzDataLakeStoreItem	Remove-AzDataLakeGen2Item	
Set-AzDataLakeStoreItemOwner Set-AzDataLakeStoreItemPermission Set-AzDataLakeStoreItemAcl	Update-AzDataLakeGen2Item	The Update-AzDataLakeGen2Item cmdlet updates a single item only, and not recursively. If want to update recursively, list items by using the Get-AzDataLakeStoreChildItem cmdlet, then pipeline to the Update-AzDataLakeGen2Item cmdlet.
Test-AzDataLakeStoreItem	Get-AzDataLakeGen2Item	The Get-AzDataLakeGen2Item cmdlet will report an error if the item doesn't exist.

## See also

- [Known issues](#)
- [Storage PowerShell cmdlets](#)

# Manage directories and files in Azure Data Lake Storage Gen2 via the Azure CLI

8/22/2022 • 4 minutes to read • [Edit Online](#)

This article shows you how to use the [Azure CLI](#) to create and manage directories and files in storage accounts that have a hierarchical namespace.

To learn about how to get, set, and update the access control lists (ACL) of directories and files, see [Use Azure CLI to manage ACLs in Azure Data Lake Storage Gen2](#).

[Samples](#) | [Give feedback](#)

## Prerequisites

- An Azure subscription. For more information, see [Get Azure free trial](#).
- A storage account that has hierarchical namespace enabled. Follow [these](#) instructions to create one.
- Azure CLI version `2.6.0` or higher.

## Ensure that you have the correct version of Azure CLI installed

1. Open the [Azure Cloud Shell](#), or if you've [installed](#) the Azure CLI locally, open a command console application such as Windows PowerShell.
2. Verify that the version of Azure CLI that have installed is `2.6.0` or higher by using the following command.

```
az --version
```

If your version of Azure CLI is lower than `2.6.0`, then install a later version. For more information, see [Install the Azure CLI](#).

## Connect to the account

1. If you're using Azure CLI locally, run the login command.

```
az login
```

If the CLI can open your default browser, it will do so and load an Azure sign-in page.

Otherwise, open a browser page at <https://aka.ms/devicelogin> and enter the authorization code displayed in your terminal. Then, sign in with your account credentials in the browser.

To learn more about different authentication methods, see [Authorize access to blob or queue data with Azure CLI](#).

2. If your identity is associated with more than one subscription, then set your active subscription to subscription of the storage account that will host your static website.

```
az account set --subscription <subscription-id>
```

Replace the `<subscription-id>` placeholder value with the ID of your subscription.

#### NOTE

The example presented in this article show Azure Active Directory (Azure AD) authorization. To learn more about authorization methods, see [Authorize access to blob or queue data with Azure CLI](#).

## Create a container

A container acts as a file system for your files. You can create one by using the `az storage fs create` command.

This example creates a container named `my-file-system`.

```
az storage fs create -n my-file-system --account-name mystorageaccount --auth-mode login
```

## Show container properties

You can print the properties of a container to the console by using the `az storage fs show` command.

```
az storage fs show -n my-file-system --account-name mystorageaccount --auth-mode login
```

## List container contents

List the contents of a directory by using the `az storage fs file list` command.

This example lists the contents of a container named `my-file-system`.

```
az storage fs file list -f my-file-system --account-name mystorageaccount --auth-mode login
```

## Delete a container

Delete a container by using the `az storage fs delete` command.

This example deletes a container named `my-file-system`.

```
az storage fs delete -n my-file-system --account-name mystorageaccount --auth-mode login
```

## Create a directory

Create a directory reference by using the `az storage fs directory create` command.

This example adds a directory named `my-directory` to a container named `my-file-system` that is located in an account named `mystorageaccount`.

```
az storage fs directory create -n my-directory -f my-file-system --account-name mystorageaccount --auth-mode login
```

## Show directory properties

You can print the properties of a directory to the console by using the `az storage fs directory show` command.

```
az storage fs directory show -n my-directory -f my-file-system --account-name mystorageaccount --auth-mode login
```

## Rename or move a directory

Rename or move a directory by using the `az storage fs directory move` command.

This example renames a directory from the name `my-directory` to the name `my-new-directory` in the same container.

```
az storage fs directory move -n my-directory -f my-file-system --new-directory "my-file-system/my-new-directory" --account-name mystorageaccount --auth-mode login
```

This example moves a directory to a container named `my-second-file-system`.

```
az storage fs directory move -n my-directory -f my-file-system --new-directory "my-second-file-system/my-new-directory" --account-name mystorageaccount --auth-mode login
```

## Delete a directory

Delete a directory by using the `az storage fs directory delete` command.

This example deletes a directory named `my-directory`.

```
az storage fs directory delete -n my-directory -f my-file-system --account-name mystorageaccount --auth-mode login
```

## Check if a directory exists

Determine if a specific directory exists in the container by using the `az storage fs directory exists` command.

This example reveals whether a directory named `my-directory` exists in the `my-file-system` container.

```
az storage fs directory exists -n my-directory -f my-file-system --account-name mystorageaccount --auth-mode login
```

## Download from a directory

Download a file from a directory by using the `az storage fs file download` command.

This example downloads a file named `upload.txt` from a directory named `my-directory`.

```
az storage fs file download -p my-directory/upload.txt -f my-file-system -d "C:\myFolder\download.txt" --account-name mystorageaccount --auth-mode login
```

## List directory contents

List the contents of a directory by using the `az storage fs file list` command.

This example lists the contents of a directory named `my-directory` that is located in the `my-file-system` container of a storage account named `mystorageaccount`.

```
az storage fs file list -f my-file-system --path my-directory --account-name mystorageaccount --auth-mode login
```

## Upload a file to a directory

Upload a file to a directory by using the `az storage fs file upload` command.

This example uploads a file named `upload.txt` to a directory named `my-directory`.

```
az storage fs file upload -s "C:\myFolder\upload.txt" -p my-directory/upload.txt -f my-file-system --account-name mystorageaccount --auth-mode login
```

## Show file properties

You can print the properties of a file to the console by using the `az storage fs file show` command.

```
az storage fs file show -p my-file.txt -f my-file-system --account-name mystorageaccount --auth-mode login
```

## Rename or move a file

Rename or move a file by using the `az storage fs file move` command.

This example renames a file from the name `my-file.txt` to the name `my-file-renamed.txt`.

```
az storage fs file move -p my-file.txt -f my-file-system --new-path my-file-system/my-file-renamed.txt --account-name mystorageaccount --auth-mode login
```

## Delete a file

Delete a file by using the `az storage fs file delete` command.

This example deletes a file named `my-file.txt`

```
az storage fs file delete -p my-directory/my-file.txt -f my-file-system --account-name mystorageaccount --auth-mode login
```

## See also

- [Samples](#)
- [Give feedback](#)
- [Known issues](#)
- [Use Azure CLI to manage ACLs in Azure Data Lake Storage Gen2](#)

# Use .NET to manage directories and files in Azure Data Lake Storage Gen2

8/22/2022 • 4 minutes to read • [Edit Online](#)

This article shows you how to use .NET to create and manage directories and files in storage accounts that have a hierarchical namespace.

To learn about how to get, set, and update the access control lists (ACL) of directories and files, see [Use .NET to manage ACLs in Azure Data Lake Storage Gen2](#).

[Package \(NuGet\)](#) | [Samples](#) | [API reference](#) | [Gen1 to Gen2 mapping](#) | [Give Feedback](#)

## Prerequisites

- An Azure subscription. See [Get Azure free trial](#).
- A storage account that has hierarchical namespace enabled. Follow [these](#) instructions to create one.

## Set up your project

To get started, install the [Azure.Storage.Files.DataLake](#) NuGet package.

For more information about how to install NuGet packages, see [Install and manage packages in Visual Studio using the NuGet Package Manager](#).

Then, add these using statements to the top of your code file.

```
using Azure;
using Azure.Storage.Files.DataLake;
using Azure.Storage.Files.DataLake.Models;
using Azure.Storage;
using System.IO;
```

## Connect to the account

To use the snippets in this article, you'll need to create a [DataLakeServiceClient](#) instance that represents the storage account.

### Connect by using an account key

This is the easiest way to connect to an account.

This example creates a [DataLakeServiceClient](#) instance by using an account key.

```

public static void GetDataLakeServiceClient(ref DataLakeServiceClient dataLakeServiceClient,
 string accountName, string accountKey)
{
 StorageSharedKeyCredential sharedKeyCredential =
 new StorageSharedKeyCredential(accountName, accountKey);

 string dfsUri = "https://" + accountName + ".dfs.core.windows.net";

 dataLakeServiceClient = new DataLakeServiceClient
 (new Uri(dfsUri), sharedKeyCredential);
}

```

## Connect by using Azure Active Directory (Azure AD)

You can use the [Azure identity client library for .NET](#) to authenticate your application with Azure AD.

This example creates a [DataLakeServiceClient](#) instance by using a client ID, a client secret, and a tenant ID. To get these values, see [Acquire a token from Azure AD for authorizing requests from a client application](#).

```

public static void GetDataLakeServiceClient(ref DataLakeServiceClient dataLakeServiceClient,
 String accountName, String clientID, string clientSecret, string tenantID)
{

 TokenCredential credential = new ClientSecretCredential(
 tenantID, clientID, clientSecret, new TokenCredentialOptions());

 string dfsUri = "https://" + accountName + ".dfs.core.windows.net";

 dataLakeServiceClient = new DataLakeServiceClient(new Uri(dfsUri), credential);
}

```

### NOTE

For more examples, see the [Azure identity client library for .NET](#) documentation..

## Create a container

A container acts as a file system for your files. You can create one by calling the [DataLakeServiceClient.CreateFileSystem](#) method.

This example creates a container named `my-file-system`.

```

public async Task<DataLakeFileSystemClient> CreateFileSystem
 (DataLakeServiceClient serviceClient)
{
 return await serviceClient.CreateFileSystemAsync("my-file-system");
}

```

## Create a directory

Create a directory reference by calling the [DataLakeFileSystemClient.CreateDirectoryAsync](#) method.

This example adds a directory named `my-directory` to a container, and then adds a sub-directory named `my-subdirectory`.

```

public async Task<DataLakeDirectoryClient> CreateDirectory
 (DataLakeServiceClient serviceClient, string fileSystemName)
{
 DataLakeFileSystemClient fileSystemClient =
 serviceClient.GetFileSystemClient(fileSystemName);

 DataLakeDirectoryClient directoryClient =
 await fileSystemClient.CreateDirectoryAsync("my-directory");

 return await directoryClient.CreateSubDirectoryAsync("my-subdirectory");
}

```

## Rename or move a directory

Rename or move a directory by calling the [DataLakeDirectoryClient.RenameAsync](#) method. Pass the path of the desired directory a parameter.

This example renames a sub-directory to the name `my-subdirectory-renamed`.

```

public async Task<DataLakeDirectoryClient>
 RenameDirectory(DataLakeFileSystemClient fileSystemClient)
{
 DataLakeDirectoryClient directoryClient =
 fileSystemClient.GetDirectoryClient("my-directory/my-subdirectory");

 return await directoryClient.RenameAsync("my-directory/my-subdirectory-renamed");
}

```

This example moves a directory named `my-subdirectory-renamed` to a sub-directory of a directory named `my-directory-2`.

```

public async Task<DataLakeDirectoryClient> MoveDirectory
 (DataLakeFileSystemClient fileSystemClient)
{
 DataLakeDirectoryClient directoryClient =
 fileSystemClient.GetDirectoryClient("my-directory/my-subdirectory-renamed");

 return await directoryClient.RenameAsync("my-directory-2/my-subdirectory-renamed");
}

```

## Delete a directory

Delete a directory by calling the [DataLakeDirectoryClient.Delete](#) method.

This example deletes a directory named `my-directory`.

```

public void DeleteDirectory(DataLakeFileSystemClient fileSystemClient)
{
 DataLakeDirectoryClient directoryClient =
 fileSystemClient.GetDirectoryClient("my-directory");

 directoryClient.Delete();
}

```

## Upload a file to a directory

First, create a file reference in the target directory by creating an instance of the [DataLakeFileClient](#) class. Upload

a file by calling the [DataLakeFileClient.AppendAsync](#) method. Make sure to complete the upload by calling the [DataLakeFileClient.FlushAsync](#) method.

This example uploads a text file to a directory named `my-directory`.

```
public async Task UploadFile(DataLakeFileSystemClient fileSystemClient)
{
 DataLakeDirectoryClient directoryClient =
 fileSystemClient.GetDirectoryClient("my-directory");

 DataLakeFileClient fileClient = await directoryClient.CreateFileAsync("uploaded-file.txt");

 FileStream fileStream =
 File.OpenRead("C:\\Users\\contoso\\Temp\\file-to-upload.txt");

 long fileSize = fileStream.Length;

 await fileClient.AppendAsync(fileStream, offset: 0);

 await fileClient.FlushAsync(position: fileSize);
}
```

#### TIP

If your file size is large, your code will have to make multiple calls to the [DataLakeFileClient.AppendAsync](#). Consider using the [DataLakeFileClient.UploadAsync](#) method instead. That way, you can upload the entire file in a single call.

See the next section for an example.

## Upload a large file to a directory

Use the [DataLakeFileClient.UploadAsync](#) method to upload large files without having to make multiple calls to the [DataLakeFileClient.AppendAsync](#) method.

```
public async Task UploadFileBulk(DataLakeFileSystemClient fileSystemClient)
{
 DataLakeDirectoryClient directoryClient =
 fileSystemClient.GetDirectoryClient("my-directory");

 DataLakeFileClient fileClient = directoryClient.GetFileClient("uploaded-file.txt");

 FileStream fileStream =
 File.OpenRead("C:\\Users\\contoso\\file-to-upload.txt");

 await fileClient.UploadAsync(fileStream);

}
```

## Download from a directory

First, create a [DataLakeFileClient](#) instance that represents the file that you want to download. Use the [DataLakeFileClient.ReadAsync](#) method, and parse the return value to obtain a [Stream](#) object. Use any .NET file processing API to save bytes from the stream to a file.

This example uses a [BinaryReader](#) and a [FileStream](#) to save bytes to a file.

```

public async Task DownloadFile(DataLakeFileSystemClient fileSystemClient)
{
 DataLakeDirectoryClient directoryClient =
 fileSystemClient.GetDirectoryClient("my-directory");

 DataLakeFileClient fileClient =
 directoryClient.GetFileClient("my-image.png");

 Response<FileDownloadInfo> downloadResponse = await fileClient.ReadAsync();

 BinaryReader reader = new BinaryReader(downloadResponse.Value.Content);

 FileStream fileStream =
 File.OpenWrite("C:\\Users\\contoso\\my-image-downloaded.png");

 int bufferSize = 4096;

 byte[] buffer = new byte[bufferSize];

 int count;

 while ((count = reader.Read(buffer, 0, buffer.Length)) != 0)
 {
 fileStream.Write(buffer, 0, count);
 }

 await fileStream.FlushAsync();

 fileStream.Close();
}

```

## List directory contents

List directory contents by calling the [FileSystemClient.GetPathsAsync](#) method, and then enumerating through the results.

This example, prints the names of each file that is located in a directory named `my-directory`.

```

public async Task ListFilesInDirectory(DataLakeFileSystemClient fileSystemClient)
{
 IAsyncEnumerator<PathItem> enumerator =
 fileSystemClient.GetPathsAsync("my-directory").GetAsyncEnumerator();

 await enumerator.MoveNextAsync();

 PathItem item = enumerator.Current;

 while (item != null)
 {
 Console.WriteLine(item.Name);

 if (!await enumerator.MoveNextAsync())
 {
 break;
 }

 item = enumerator.Current;
 }
}

```

## See also

- [API reference documentation](#)
- [Package \(NuGet\)](#)
- [Samples](#)
- [Gen1 to Gen2 mapping](#)
- [Known issues](#)
- [Give Feedback](#)

# Use Java to manage directories and files in Azure Data Lake Storage Gen2

8/22/2022 • 4 minutes to read • [Edit Online](#)

This article shows you how to use Java to create and manage directories and files in storage accounts that have a hierarchical namespace.

To learn about how to get, set, and update the access control lists (ACL) of directories and files, see [Use Java to manage ACLs in Azure Data Lake Storage Gen2](#).

[Package \(Maven\)](#) | [Samples](#) | [API reference](#) | [Gen1 to Gen2 mapping](#) | [Give Feedback](#)

## Prerequisites

- An Azure subscription. For more information, see [Get Azure free trial](#).
- A storage account that has hierarchical namespace enabled. Follow [these](#) instructions to create one.

## Set up your project

To get started, open [this page](#) and find the latest version of the Java library. Then, open the *pom.xml* file in your text editor. Add a dependency element that references that version.

If you plan to authenticate your client application by using Azure Active Directory (Azure AD), then add a dependency to the Azure Secret Client Library. For more information, see [Adding the Secret Client Library package to your project](#).

Next, add these imports statements to your code file.

```
import com.azure.storage.common.StorageSharedKeyCredential;
import com.azure.storage.file.datalake.DataLakeDirectoryClient;
import com.azure.storage.file.datalake.DataLakeFileClient;
import com.azure.storage.file.datalake.DataLakeFileSystemClient;
import com.azure.storage.file.datalake.DataLakeServiceClient;
import com.azure.storage.file.datalake.DataLakeServiceClientBuilder;
import com.azure.storage.file.datalake.models.ListPathsOptions;
import com.azure.storage.file.datalake.models.PathItem;
import com.azure.storage.file.datalake.models.AccessControlChangeCounters;
import com.azure.storage.file.datalake.models.AccessControlChangeResult;
import com.azure.storage.file.datalake.models.AccessControlType;
import com.azure.storage.file.datalake.models.PathAccessControl;
import com.azure.storage.file.datalake.models.PathAccessControlEntry;
import com.azure.storage.file.datalake.models.PathPermissions;
import com.azure.storage.file.datalake.models.PathRemoveAccessControlEntry;
import com.azure.storage.file.datalake.models.RolePermissions;
import com.azure.storage.file.datalake.options.PathSetAccessControlRecursiveOptions;
```

## Connect to the account

To use the snippets in this article, you'll need to create a `DataLakeServiceClient` instance that represents the storage account.

### Connect by using an account key

This is the easiest way to connect to an account.

This example creates a **DataLakeServiceClient** instance by using an account key.

```
static public DataLakeServiceClient GetDataLakeServiceClient
(String accountName, String accountKey){

 StorageSharedKeyCredential sharedKeyCredential =
 new StorageSharedKeyCredential(accountName, accountKey);

 DataLakeServiceClientBuilder builder = new DataLakeServiceClientBuilder();

 builder.credential(sharedKeyCredential);
 builder.endpoint("https://" + accountName + ".dfs.core.windows.net");

 return builder.buildClient();
}
```

## Connect by using Azure Active Directory (Azure AD)

You can use the [Azure identity client library for Java](#) to authenticate your application with Azure AD.

This example creates a **DataLakeServiceClient** instance by using a client ID, a client secret, and a tenant ID. To get these values, see [Acquire a token from Azure AD for authorizing requests from a client application](#).

```
static public DataLakeServiceClient GetDataLakeServiceClient
(String accountName, String clientId, String ClientSecret, String tenantID){

 String endpoint = "https://" + accountName + ".dfs.core.windows.net";

 ClientSecretCredential clientSecretCredential = new ClientSecretCredentialBuilder()
 .clientId(clientId)
 .clientSecret(ClientSecret)
 .tenantId(tenantID)
 .build();

 DataLakeServiceClientBuilder builder = new DataLakeServiceClientBuilder();
 return builder.credential(clientSecretCredential).endpoint(endpoint).buildClient();
}
```

### NOTE

For more examples, see the [Azure identity client library for Java](#) documentation.

## Create a container

A container acts as a file system for your files. You can create one by calling the **DataLakeServiceClient.createFileSystem** method.

This example creates a container named `my-file-system`.

```
public DataLakeFileSystemClient CreateFileSystem
(DataLakeServiceClient serviceClient){

 return serviceClient.createFileSystem("my-file-system");
}
```

## Create a directory

Create a directory reference by calling the **DataLakeFileSystemClient.createDirectory** method.

This example adds a directory named `my-directory` to a container, and then adds a sub-directory named `my-subdirectory`.

```
public DataLakeDirectoryClient CreateDirectory
(DataLakeServiceClient serviceClient, String fileSystemName){

 DataLakeFileSystemClient fileSystemClient =
 serviceClient.getFileSystemClient(fileSystemName);

 DataLakeDirectoryClient directoryClient =
 fileSystemClient.createDirectory("my-directory");

 return directoryClient.createSubdirectory("my-subdirectory");
}
```

## Rename or move a directory

Rename or move a directory by calling the `DataLakeDirectoryClient.rename` method. Pass the path of the desired directory a parameter.

This example renames a sub-directory to the name `my-subdirectory-renamed`.

```
public DataLakeDirectoryClient
RenameDirectory(DataLakeFileSystemClient fileSystemClient){

 DataLakeDirectoryClient directoryClient =
 fileSystemClient.getDirectoryClient("my-directory/my-subdirectory");

 return directoryClient.rename(fileSystemClient.getFileSystemName(),"my-subdirectory-renamed");
}
```

This example moves a directory named `my-subdirectory-renamed` to a sub-directory of a directory named `my-directory-2`.

```
public DataLakeDirectoryClient MoveDirectory
(DataLakeFileSystemClient fileSystemClient){

 DataLakeDirectoryClient directoryClient =
 fileSystemClient.getDirectoryClient("my-directory/my-subdirectory-renamed");

 return directoryClient.rename(fileSystemClient.getFileSystemName(),"my-directory-2/my-subdirectory-
renamed");
}
```

## Delete a directory

Delete a directory by calling the `DataLakeDirectoryClient.deleteWithResponse` method.

This example deletes a directory named `my-directory`.

```
public void DeleteDirectory(DataLakeFileSystemClient fileSystemClient){

 DataLakeDirectoryClient directoryClient =
 fileSystemClient.getDirectoryClient("my-directory");

 directoryClient.deleteWithResponse(true, null, null, null);
}
```

## Upload a file to a directory

First, create a file reference in the target directory by creating an instance of the **DataLakeFileClient** class. Upload a file by calling the **DataLakeFileClient.append** method. Make sure to complete the upload by calling the **DataLakeFileClient.FlushAsync** method.

This example uploads a text file to a directory named `my-directory`.

```
public void UploadFile(DataLakeFileSystemClient fileSystemClient)
 throws FileNotFoundException{

 DataLakeDirectoryClient directoryClient =
 fileSystemClient.getDirectoryClient("my-directory");

 DataLakeFileClient fileClient = directoryClient.createFile("uploaded-file.txt");

 File file = new File("C:\\\\Users\\\\constoso\\\\mytestfile.txt");

 // InputStream targetStream = new FileInputStream(file);
 // InputStream targetStream = new BufferedInputStream(new FileInputStream(file));

 long fileSize = file.length();

 fileClient.append(targetStream, 0, fileSize);

 fileClient.flush(fileSize);
}
```

### TIP

If your file size is large, your code will have to make multiple calls to the **DataLakeFileClient.append** method. Consider using the **DataLakeFileClient.uploadFromFile** method instead. That way, you can upload the entire file in a single call.

See the next section for an example.

## Upload a large file to a directory

Use the **DataLakeFileClient.uploadFromFile** method to upload large files without having to make multiple calls to the **DataLakeFileClient.append** method.

```
public void UploadFileBulk(DataLakeFileSystemClient fileSystemClient)
 throws FileNotFoundException{

 DataLakeDirectoryClient directoryClient =
 fileSystemClient.getDirectoryClient("my-directory");

 DataLakeFileClient fileClient = directoryClient.getFileClient("uploaded-file.txt");

 fileClient.uploadFromFile("C:\\\\Users\\\\contoso\\\\mytestfile.txt");

}
```

## Download from a directory

First, create a **DataLakeFileClient** instance that represents the file that you want to download. Use the **DataLakeFileClient.read** method to read the file. Use any Java file processing API to save bytes from the stream to a file.

```
public void DownloadFile(DataLakeFileSystemClient fileSystemClient)
 throws FileNotFoundException, java.io.IOException{

 DataLakeDirectoryClient directoryClient =
 fileSystemClient.getDirectoryClient("my-directory");

 DataLakeFileClient fileClient =
 directoryClient.getFileClient("uploaded-file.txt");

 File file = new File("C:\\\\Users\\\\contoso\\\\downloadedFile.txt");

 OutputStream targetStream = new FileOutputStream(file);

 fileClient.read(targetStream);

 targetStream.close();

}
```

## List directory contents

This example prints the names of each file that is located in a directory named `my-directory`.

```
public void ListFilesInDirectory(DataLakeFileSystemClient fileSystemClient){

 ListPathsOptions options = new ListPathsOptions();
 options.setPath("my-directory");

 PagedIterable<PathItem> pagedIterable =
 fileSystemClient.listPaths(options, null);

 java.util.Iterator<PathItem> iterator = pagedIterable.iterator();

 PathItem item = iterator.next();

 while (item != null)
 {
 System.out.println(item.getName());

 if (!iterator.hasNext())
 {
 break;
 }

 item = iterator.next();
 }
}
```

## See also

- [API reference documentation](#)
- [Package \(Maven\)](#)
- [Samples](#)
- [Gen1 to Gen2 mapping](#)
- [Known issues](#)
- [Give Feedback](#)

# Use Python to manage directories and files in Azure Data Lake Storage Gen2

8/22/2022 • 4 minutes to read • [Edit Online](#)

This article shows you how to use Python to create and manage directories and files in storage accounts that have a hierarchical namespace.

To learn about how to get, set, and update the access control lists (ACL) of directories and files, see [Use Python to manage ACLs in Azure Data Lake Storage Gen2](#).

[Package \(Python Package Index\)](#) | [Samples](#) | [API reference](#) | [Gen1 to Gen2 mapping](#) | [Give Feedback](#)

## Prerequisites

- An Azure subscription. See [Get Azure free trial](#).
- A storage account that has hierarchical namespace enabled. Follow [these](#) instructions to create one.

## Set up your project

Install the Azure Data Lake Storage client library for Python by using [pip](#).

```
pip install azure-storage-file-datalake
```

Add these import statements to the top of your code file.

```
import os, uuid, sys
from azure.storage.filedatalake import DataLakeServiceClient
from azure.core._match_conditions import MatchConditions
from azure.storage.filedatalake._models import ContentSettings
```

## Connect to the account

To use the snippets in this article, you'll need to create a `DataLakeServiceClient` instance that represents the storage account.

### Connect by using an account key

This is the easiest way to connect to an account.

This example creates a `DataLakeServiceClient` instance by using an account key.

```
def initialize_storage_account(storage_account_name, storage_account_key):

 try:
 global service_client

 service_client = DataLakeServiceClient(account_url="{}://{}.dfs.core.windows.net".format(
 "https", storage_account_name), credential=storage_account_key)

 except Exception as e:
 print(e)
```

- Replace the `storage_account_name` placeholder value with the name of your storage account.
- Replace the `storage_account_key` placeholder value with your storage account access key.

## Connect by using Azure Active Directory (Azure AD)

You can use the [Azure identity client library for Python](#) to authenticate your application with Azure AD.

This example creates a `DataLakeServiceClient` instance by using a client ID, a client secret, and a tenant ID. To get these values, see [Acquire a token from Azure AD for authorizing requests from a client application](#).

```
def initialize_storage_account_ad(storage_account_name, client_id, client_secret, tenant_id):

 try:
 global service_client

 credential = ClientSecretCredential(tenant_id, client_id, client_secret)

 service_client = DataLakeServiceClient(account_url="{}://{}.dfs.core.windows.net".format(
 "https", storage_account_name), credential=credential)

 except Exception as e:
 print(e)
```

### NOTE

For more examples, see the [Azure identity client library for Python](#) documentation.

## Create a container

A container acts as a file system for your files. You can create one by calling the `FileSystemDataLakeServiceClient.create_file_system` method.

This example creates a container named `my-file-system`.

```
def create_file_system():
 try:
 global file_system_client

 file_system_client = service_client.create_file_system(file_system="my-file-system")

 except Exception as e:
 print(e)
```

## Create a directory

Create a directory reference by calling the `FileSystemClient.create_directory` method.

This example adds a directory named `my-directory` to a container.

```
def create_directory():
 try:
 file_system_client.create_directory("my-directory")

 except Exception as e:
 print(e)
```

## Rename or move a directory

Rename or move a directory by calling the **DataLakeDirectoryClient.rename\_directory** method. Pass the path of the desired directory a parameter.

This example renames a sub-directory to the name `my-directory-renamed`.

```
def rename_directory():
 try:

 file_system_client = service_client.get_file_system_client(file_system="my-file-system")
 directory_client = file_system_client.get_directory_client("my-directory")

 new_dir_name = "my-directory-renamed"
 directory_client.rename_directory(new_name=directory_client.file_system_name + '/' + new_dir_name)

 except Exception as e:
 print(e)
```

## Delete a directory

Delete a directory by calling the **DataLakeDirectoryClient.delete\_directory** method.

This example deletes a directory named `my-directory`.

```
def delete_directory():
 try:
 file_system_client = service_client.get_file_system_client(file_system="my-file-system")
 directory_client = file_system_client.get_directory_client("my-directory")

 directory_client.delete_directory()
 except Exception as e:
 print(e)
```

## Upload a file to a directory

First, create a file reference in the target directory by creating an instance of the **DataLakeFileClient** class.

Upload a file by calling the **DataLakeFileClient.append\_data** method. Make sure to complete the upload by calling the **DataLakeFileClient.flush\_data** method.

This example uploads a text file to a directory named `my-directory`.

```
def upload_file_to_directory():
 try:

 file_system_client = service_client.get_file_system_client(file_system="my-file-system")

 directory_client = file_system_client.get_directory_client("my-directory")

 file_client = directory_client.create_file("uploaded-file.txt")
 local_file = open("C:\\\\file-to-upload.txt",'r')

 file_contents = local_file.read()

 file_client.append_data(data=file_contents, offset=0, length=len(file_contents))

 file_client.flush_data(len(file_contents))

 except Exception as e:
 print(e)
```

#### TIP

If your file size is large, your code will have to make multiple calls to the `DataLakeFileClient.append_data` method. Consider using the `DataLakeFileClient.upload_data` method instead. That way, you can upload the entire file in a single call.

## Upload a large file to a directory

Use the `DataLakeFileClient.upload_data` method to upload large files without having to make multiple calls to the `DataLakeFileClient.append_data` method.

```
def upload_file_to_directory_bulk():
 try:

 file_system_client = service_client.get_file_system_client(file_system="my-file-system")

 directory_client = file_system_client.get_directory_client("my-directory")

 file_client = directory_client.get_file_client("uploaded-file.txt")

 local_file = open("C:\\\\file-to-upload.txt",'r')

 file_contents = local_file.read()

 file_client.upload_data(file_contents, overwrite=True)

 except Exception as e:
 print(e)
```

## Download from a directory

Open a local file for writing. Then, create a `DataLakeFileClient` instance that represents the file that you want to download. Call the `DataLakeFileClient.read_file` to read bytes from the file and then write those bytes to the local file.

```
def download_file_from_directory():
 try:
 file_system_client = service_client.get_file_system_client(file_system="my-file-system")

 directory_client = file_system_client.get_directory_client("my-directory")

 local_file = open("C:\\file-to-download.txt", 'wb')

 file_client = directory_client.get_file_client("uploaded-file.txt")

 download = file_client.download_file()

 downloaded_bytes = download.readall()

 local_file.write(downloaded_bytes)

 local_file.close()

 except Exception as e:
 print(e)
```

## List directory contents

List directory contents by calling the `FileSystemClient.get_paths` method, and then enumerating through the results.

This example, prints the path of each subdirectory and file that is located in a directory named `my-directory`.

```
def list_directory_contents():
 try:

 file_system_client = service_client.get_file_system_client(file_system="my-file-system")

 paths = file_system_client.get_paths(path="my-directory")

 for path in paths:
 print(path.name + '\n')

 except Exception as e:
 print(e)
```

## See also

- [API reference documentation](#)
- [Package \(Python Package Index\)](#)
- [Samples](#)
- [Gen1 to Gen2 mapping](#)
- [Known issues](#)
- [Give Feedback](#)

# Use JavaScript SDK in Node.js to manage directories and files in Azure Data Lake Storage Gen2

8/22/2022 • 4 minutes to read • [Edit Online](#)

This article shows you how to use Node.js to create and manage directories and files in storage accounts that have a hierarchical namespace.

To learn about how to get, set, and update the access control lists (ACL) of directories and files, see [Use JavaScript SDK in Node.js to manage ACLs in Azure Data Lake Storage Gen2](#).

[Package \(Node Package Manager\)](#) | [Samples](#) | [Give Feedback](#)

## Prerequisites

- An Azure subscription. For more information, see [Get Azure free trial](#).
- A storage account that has hierarchical namespace enabled. Follow [these](#) instructions to create one.
- If you are using this package in a Node.js application, you'll need Node.js 8.0.0 or higher.

## Set up your project

Install Data Lake client library for JavaScript by opening a terminal window, and then typing the following command.

```
npm install @azure/storage-file-datalake
```

Import the `storage-file-datalake` package by placing this statement at the top of your code file.

```
const {
 AzureStorageDataLake,
 DataLakeServiceClient,
 StorageSharedKeyCredential
} = require("@azure/storage-file-datalake");
```

## Connect to the account

To use the snippets in this article, you'll need to create a `DataLakeServiceClient` instance that represents the storage account.

### Connect by using an account key

This is the easiest way to connect to an account.

This example creates a `DataLakeServiceClient` instance by using an account key.

```
function GetDataLakeServiceClient(accountName, accountKey) {

 const sharedKeyCredential =
 new StorageSharedKeyCredential(accountName, accountKey);

 const datalakeServiceClient = new DataLakeServiceClient(
 `https://${accountName}.dfs.core.windows.net`, sharedKeyCredential);

 return datalakeServiceClient;
}
```

#### NOTE

This method of authorization works only for Node.js applications. If you plan to run your code in a browser, you can authorize by using Azure Active Directory (Azure AD).

### Connect by using Azure Active Directory (Azure AD)

You can use the [Azure identity client library for JS](#) to authenticate your application with Azure AD.

This example creates a **DataLakeServiceClient** instance by using a client ID, a client secret, and a tenant ID. To get these values, see [Acquire a token from Azure AD for authorizing requests from a client application](#).

```
function GetDataLakeServiceClientAD(accountName, clientId, clientSecret, tenantID) {

 const credential = new ClientSecretCredential(tenantID, clientId, clientSecret);

 const datalakeServiceClient = new DataLakeServiceClient(
 `https://${accountName}.dfs.core.windows.net`, credential);

 return datalakeServiceClient;
}
```

#### NOTE

For more examples, see the [Azure identity client library for JS](#) documentation.

## Create a container

A container acts as a file system for your files. You can create one by getting a **FileSystemClient** instance, and then calling the **FileSystemClient.Create** method.

This example creates a container named `my-file-system`.

```
async function CreateFileSystem(datalakeServiceClient) {

 const fileName = "my-file-system";

 const fileSystemClient = datalakeServiceClient.getFileSystemClient(fileName);

 const createResponse = await fileSystemClient.create();
}
```

## Create a directory

Create a directory reference by getting a **DirectoryClient** instance, and then calling the **DirectoryClient.create** method.

This example adds a directory named `my-directory` to a container.

```
async function CreateDirectory(fileSystemClient) {

 const directoryClient = fileSystemClient.getDirectoryClient("my-directory");

 await directoryClient.create();

}
```

## Rename or move a directory

Rename or move a directory by calling the **DirectoryClient.rename** method. Pass the path of the desired directory a parameter.

This example renames a sub-directory to the name `my-directory-renamed`.

```
async function RenameDirectory(fileSystemClient) {

 const directoryClient = fileSystemClient.getDirectoryClient("my-directory");
 await directoryClient.move("my-directory-renamed");

}
```

This example moves a directory named `my-directory-renamed` to a sub-directory of a directory named `my-directory-2`.

```
async function MoveDirectory(fileSystemClient) {

 const directoryClient = fileSystemClient.getDirectoryClient("my-directory-renamed");
 await directoryClient.move("my-directory-2/my-directory-renamed");

}
```

## Delete a directory

Delete a directory by calling the **DirectoryClient.delete** method.

This example deletes a directory named `my-directory`.

```
async function DeleteDirectory(fileSystemClient) {

 const directoryClient = fileSystemClient.getDirectoryClient("my-directory");
 await directoryClient.delete();

}
```

## Upload a file to a directory

First, read a file. This example uses the Node.js `fs` module. Then, create a file reference in the target directory by creating a **FileClient** instance, and then calling the **FileClient.create** method. Upload a file by calling the

**FileClient.append** method. Make sure to complete the upload by calling the **FileClient.flush** method.

This example uploads a text file to a directory named `my-directory`.

```
async function UploadFile(fileSystemClient) {

 const fs = require('fs')

 var content = "";

 fs.readFile('mytestfile.txt', (err, data) => {
 if (err) throw err;

 content = data.toString();
 })

 const fileClient = fileSystemClient.getFileClient("my-directory/uploaded-file.txt");
 await fileClient.create();
 await fileClient.append(content, 0, content.length);
 await fileClient.flush(content.length);

}
```

## Download from a directory

First, create a **FileSystemClient** instance that represents the file that you want to download. Use the **FileSystemClient.read** method to read the file. Then, write the file. This example uses the Node.js `fs` module to do that.

### NOTE

This method of downloading a file works only for Node.js applications. If you plan to run your code in a browser, see the [Azure Storage File Data Lake client library for JavaScript](#) readme file for an example of how to do this in a browser.

```

async function DownloadFile(fileSystemClient) {

 const fileClient = fileSystemClient.getFileClient("my-directory/uploaded-file.txt");

 const downloadResponse = await fileClient.read();

 const downloaded = await streamToString(downloadResponse.readableStreamBody);

 async function streamToString(readableStream) {
 return new Promise((resolve, reject) => {
 const chunks = [];
 readableStream.on("data", (data) => {
 chunks.push(data.toString());
 });
 readableStream.on("end", () => {
 resolve(chunks.join(""));
 });
 readableStream.on("error", reject);
 });
 }

 const fs = require('fs');

 fs.writeFile('mytestfiledownloaded.txt', downloaded, (err) => {
 if (err) throw err;
 });
}

```

## List directory contents

This example, prints the names of each directory and file that is located in a directory named `my-directory`.

```

async function ListFilesInDirectory(fileSystemClient) {

let i = 1;

let iter = await fileSystemClient.listPaths({path: "my-directory", recursive: true});

for await (const path of iter) {

 console.log(`Path ${i++}: ${path.name}, is directory: ${path.isDirectory}`);
}

}

```

## See also

- [Package \(Node Package Manager\)](#)
- [Samples](#)
- [Give Feedback](#)

# Filter data by using Azure Data Lake Storage query acceleration

8/22/2022 • 9 minutes to read • [Edit Online](#)

This article shows you how to use query acceleration to retrieve a subset of data from your storage account.

Query acceleration enables applications and analytics frameworks to dramatically optimize data processing by retrieving only the data that they require to perform a given operation. To learn more, see [Azure Data Lake Storage Query Acceleration](#).

## Prerequisites

- To access Azure Storage, you'll need an Azure subscription. If you don't already have a subscription, create a [free account](#) before you begin.
- A **general-purpose v2** storage account. see [Create a storage account](#).
- Choose a tab to view any SDK-specific prerequisites.
  - [PowerShell](#)
  - [.NET v12 SDK](#)
  - [Java v12 SDK](#)
  - [Python v12 SDK](#)
  - [Node.js v12 SDK](#)

Not applicable

## Set up your environment

### Step 1: Install packages

- [PowerShell](#)
- [.NET v12 SDK](#)
- [Java v12 SDK](#)
- [Python v12 SDK](#)
- [Node.js v12 SDK](#)

Install the Az module version 4.6.0 or higher.

```
Install-Module -Name Az -Repository PSGallery -Force
```

To update from an older version of Az, run the following command:

```
Update-Module -Name Az
```

### Step 2: Add statements

- [PowerShell](#)
- [.NET v12 SDK](#)

- [Java v12 SDK](#)
- [Python v12 SDK](#)
- [Node.js v12 SDK](#)

Not applicable

## Retrieve data by using a filter

You can use SQL to specify the row filter predicates and column projections in a query acceleration request. The following code queries a CSV file in storage and returns all rows of data where the third column matches the value `Hemingway, Ernest`.

- In the SQL query, the keyword `BlobStorage` is used to denote the file that is being queried.
- Column references are specified as `_N` where the first column is `_1`. If the source file contains a header row, then you can refer to columns by the name that is specified in the header row.
- [PowerShell](#)
- [.NET v12 SDK](#)
- [Java v12 SDK](#)
- [Python v12 SDK](#)
- [Node.js v12 SDK](#)

```
Function Get-QueryCsv($ctx, $container, $blob, $query, $hasheaders) {
 $tempfile = New-TemporaryFile
 $informat = New-AzStorageBlobQueryConfig -AsCsv -HasHeader:$hasheaders
 Get-AzStorageBlobQueryResult -Context $ctx -Container $container -Blob $blob -InputTextConfiguration
 $informat -OutputTextConfiguration (New-AzStorageBlobQueryConfig -AsCsv -HasHeader) -ResultFile
 $tempfile.FullName -QueryString $query -Force
 Get-Content $tempfile.FullName
}

$container = "data"
$blob = "csv/csv-general/seattle-library.csv"
Get-QueryCsv $ctx $container $blob "SELECT * FROM BlobStorage WHERE _3 = 'Hemingway, Ernest, 1899-1961'"
$false
```

## Retrieve specific columns

You can scope your results to a subset of columns. That way you retrieve only the columns needed to perform a given calculation. This improves application performance and reduces cost because less data is transferred over the network.

### NOTE

The maximum number of columns that you can scope your results to is 49. If you need your results to contain more than 49 columns, then use a wildcard character (`*`) for the SELECT expression (For example: `SELECT *`).

This code retrieves only the `BibNum` column for all books in the data set. It also uses the information from the header row in the source file to reference columns in the query.

- [PowerShell](#)
- [.NET v12 SDK](#)

- [Java v12 SDK](#)
- [Python v12 SDK](#)
- [Node.js v12 SDK](#)

```
Function Get-QueryCsv($ctx, $container, $blob, $query, $hasheaders) {
 $tempfile = New-TemporaryFile
 $informat = New-AzStorageBlobQueryConfig -AsCsv -HasHeader:$hasheaders
 Get-AzStorageBlobQueryResult -Context $ctx -Container $container -Blob $blob -InputTextConfiguration
 $informat -OutputTextConfiguration (New-AzStorageBlobQueryConfig -AsCsv -HasHeader) -ResultFile
 $tempfile.FullName -QueryString $query -Force
 Get-Content $tempfile.FullName
}

$container = "data"
$blob = "csv/csv-general/seattle-library-with-headers.csv"
Get-QueryCsv $ctx $container $blob "SELECT BibNum FROM BlobStorage" $true
```

The following code combines row filtering and column projections into the same query.

- [PowerShell](#)
- [.NET v12 SDK](#)
- [Java v12 SDK](#)
- [Python v12 SDK](#)
- [Node.js v12 SDK](#)

```
Get-QueryCsv $ctx $container $blob $query $true

Function Get-QueryCsv($ctx, $container, $blob, $query, $hasheaders) {
 $tempfile = New-TemporaryFile
 $informat = New-AzStorageBlobQueryConfig -AsCsv -HasHeader:$hasheaders
 Get-AzStorageBlobQueryResult -Context $ctx -Container $container -Blob $blob -InputTextConfiguration
 $informat -OutputTextConfiguration (New-AzStorageBlobQueryConfig -AsCsv -HasHeader) -ResultFile
 $tempfile.FullName -QueryString $query -Force
 Get-Content $tempfile.FullName
}

$container = "data"
$query = "SELECT BibNum, Title, Author, ISBN, Publisher, ItemType
 FROM BlobStorage
 WHERE ItemType IN
 ('acdvd', 'cadvd', 'cadvdnf', 'calndvd', 'ccdvd', 'ccdvdnf', 'jcdvd', 'nadvd', 'nadvdnf',
 'nalndvd', 'ncdvd', 'ncdvdnf')"
```

## Next steps

- [Azure Data Lake Storage query acceleration](#)
- [Query acceleration SQL language reference](#)

# Using the HDFS CLI with Data Lake Storage Gen2

8/22/2022 • 2 minutes to read • [Edit Online](#)

You can access and manage the data in your storage account by using a command line interface just as you would with a [Hadoop Distributed File System \(HDFS\)](#). This article provides some examples that will help you get started.

HDInsight provides access to the distributed container that is locally attached to the compute nodes. You can access this container by using the shell that directly interacts with the HDFS and the other file systems that Hadoop supports.

For more information on HDFS CLI, see the [official documentation](#) and the [HDFS Permissions Guide](#)

## NOTE

If you're using Azure Databricks instead of HDInsight, and you want to interact with your data by using a command line interface, you can use the Databricks CLI to interact with the Databricks file system. See [Databricks CLI](#).

## Use the HDFS CLI with an HDInsight Hadoop cluster on Linux

First, establish [remote access to services](#). If you pick [SSH](#) the sample PowerShell code would look as follows:

```
#Connect to the cluster via SSH.
ssh sshuser@clusternamespace-ssh.azurehdinsight.net
#Execute basic HDFS commands. Display the hierarchy.
hdfs dfs -ls /
#Create a sample directory.
hdfs dfs -mkdir /samplefolder
```

The connection string can be found at the "SSH + Cluster login" section of the HDInsight cluster blade in Azure portal. SSH credentials were specified at the time of the cluster creation.

## IMPORTANT

HDInsight cluster billing starts after a cluster is created and stops when the cluster is deleted. Billing is pro-rated per minute, so you should always delete your cluster when it is no longer in use. To learn how to delete a cluster, see our [article on the topic](#). However, data stored in a storage account with Data Lake Storage Gen2 enabled persists even after an HDInsight cluster is deleted.

## Create a container

```
hdfs dfs -D "fs.azure.createRemoteFileSystemDuringInitialization=true" -ls abfs://<container-name>@<storage-account-name>.dfs.core.windows.net/
```

- Replace the `<container-name>` placeholder with the name that you want to give your container.
- Replace the `<storage-account-name>` placeholder with the name of your storage account.

## Get a list of files or directories

```
hdfs dfs -ls <path>
```

Replace the <path> placeholder with the URI of the container or container folder.

For example: `hdfs dfs -ls abfs://my-file-system@mystorageaccount.dfs.core.windows.net/my-directory-name`

## Create a directory

```
hdfs dfs -mkdir [-p] <path>
```

Replace the <path> placeholder with the root container name or a folder within your container.

For example: `hdfs dfs -mkdir abfs://my-file-system@mystorageaccount.dfs.core.windows.net/`

## Delete a file or directory

```
hdfs dfs -rm <path>
```

Replace the <path> placeholder with the URI of the file or folder that you want to delete.

For example:

```
hdfs dfs -rmdir abfs://my-file-system@mystorageaccount.dfs.core.windows.net/my-directory-name/my-file-name
```

## Display the Access Control Lists (ACLs) of files and directories

```
hdfs dfs -getfacl [-R] <path>
```

Example:

```
hdfs dfs -getfacl -R /dir
```

See [getfacl](#)

## Set ACLs of files and directories

```
hdfs dfs -setfacl [-R] [-b|-k -m|-x <acl_spec> <path>]|[--set <acl_spec> <path>]
```

Example:

```
hdfs dfs -setfacl -m user:hadoop:rw- /file
```

See [setfacl](#)

## Change the owner of files

```
hdfs dfs -chown [-R] <new_owner>:<users_group> <URI>
```

See [chown](#)

## Change group association of files

```
hdfs dfs -chgrp [-R] <group> <URI>
```

See [chgrp](#)

## Change the permissions of files

```
hdfs dfs -chmod [-R] <mode> <URI>
```

See [chmod](#)

You can view the complete list of commands on the [Apache Hadoop 2.4.1 File System Shell Guide Website](#).

## Next steps

- [Use an Azure Data Lake Storage Gen2 capable account in Azure Databricks](#)
- [Learn about access control lists on files and directories](#)

# Tutorials that use Azure services with Azure Data Lake Storage Gen2

8/22/2022 • 2 minutes to read • [Edit Online](#)

This article contains links to tutorials that show you how to use various Azure services with Data Lake Storage Gen2.

## List of tutorials

AZURE SERVICE	STEP-BY-STEP GUIDE
Azure Synapse Analytics	<a href="#">Get Started with Azure Synapse Analytics</a>
Azure Data Factory	<a href="#">Load data into Azure Data Lake Storage Gen2 with Azure Data Factory</a>
Azure Databricks	<a href="#">Use with Azure Databricks</a>
Azure Databricks	<a href="#">Extract, transform, and load data by using Azure Databricks</a>
Azure Databricks	<a href="#">Access Data Lake Storage Gen2 data with Azure Databricks using Spark</a>
Azure Event Grid	<a href="#">Implement the data lake capture pattern to update a Databricks Delta table</a>
Azure Machine Learning	<a href="#">Access data in Azure storage services</a>
Azure Data Box	<a href="#">Use Azure Data Box to migrate data from an on-premises HDFS store to Azure Storage</a>
HDInsight	<a href="#">Use Azure Data Lake Storage Gen2 with Azure HDInsight clusters</a>
HDInsight	<a href="#">Extract, transform, and load data by using Apache Hive on Azure HDInsight</a>
Power BI	<a href="#">Analyze data in Data Lake Storage Gen2 using Power BI</a>
Azure Data Explorer	<a href="#">Query data in Azure Data Lake using Azure Data Explorer</a>
Azure Cognitive Search	<a href="#">Index and search Azure Data Lake Storage Gen2 documents (preview)</a>

### NOTE

This table doesn't reflect the complete list of Azure services that support Data Lake Storage Gen2. To see a list of supported Azure services, their level of support, see [Azure services that support Azure Data Lake Storage Gen2](#). To see how services organized into categories such as ingest, download, process, and visualize, see [Ingest, process, and analyze](#).

## See also

[Best practices for using Azure Data Lake Storage Gen2](#)

# Query acceleration SQL language reference

8/22/2022 • 10 minutes to read • [Edit Online](#)

Query acceleration supports an ANSI SQL-like language for expressing queries over blob contents. The query acceleration SQL dialect is a subset of ANSI SQL, with a limited set of supported data types, operators, etc., but it also expands on ANSI SQL to support queries over hierarchical semi-structured data formats such as JSON.

## SELECT Syntax

The only SQL statement supported by query acceleration is the SELECT statement. This example returns every row for which expression returns true.

```
SELECT * FROM table [WHERE expression] [LIMIT limit]
```

For CSV-formatted data, *table* must be `BlobStorage`. This means that the query will run against whichever blob was specified in the REST call. For JSON-formatted data, *table* is a "table descriptor." See the [Table Descriptors](#) section of this article.

In the following example, for each row for which the WHERE *expression* returns true, this statement will return a new row that is made from evaluating each of the projection expressions.

```
SELECT expression [, expression ...] FROM table [WHERE expression] [LIMIT limit]
```

You can specify one or more specific columns as part of the SELECT expression (for example,

```
SELECT Title, Author, ISBN).
```

### NOTE

The maximum number of specific columns that you can use in the SELECT expression is 49. If you need your SELECT statement to return more than 49 columns, then use a wildcard character (`*`) for the SELECT expression (For example: `SELECT *`).

The following example returns an aggregate computation (For example: the average value of a particular column) over each of the rows for which *expression* returns true.

```
SELECT aggregate_expression FROM table [WHERE expression] [LIMIT limit]
```

The following example returns suitable offsets for splitting a CSV-formatted blob. See the [Sys.Split](#) section of this article.

```
SELECT sys.split(split_size)FROM BlobStorage
```

## Data Types

DATA TYPE	DESCRIPTION
INT	64-bit signed integer.
FLOAT	64-bit ("double-precision") floating point.
STRING	Variable-length Unicode string.
TIMESTAMP	A point in time.
BOOLEAN	True or false.

When reading values from CSV-formatted data, all values are read as strings. String values may be converted to other types using CAST expressions. Values may be implicitly cast to other types depending on context. for more info, see [Data type precedence \(Transact-SQL\)](#).

## Expressions

### Referencing fields

For JSON-formatted data, or CSV-formatted data with a header row, fields may be referenced by name. Field names can be quoted or unquoted. Quoted field names are enclosed in double-quote characters ("), may contain spaces, and are case-sensitive. Unquoted field names are case-insensitive, and may not contain any special characters.

In CSV-formatted data, fields may also be referenced by ordinal, prefixed with an underscore (\_) character. For example, the first field may be referenced as \_1, or the eleventh field may be referenced as \_11. Referencing fields by ordinal is useful for CSV-formatted data that does not contain a header row, in which case the only way to reference a particular field is by ordinal.

### Operators

The following standard SQL operators are supported:

OPERATOR	DESCRIPTION
=	Compares the equality of two expressions (a comparison operator).
!=	Tests whether one expression is not equal to another expression (a comparison operator).
<>	Compares two expressions for not equal to (a comparison operator).
<	Compares two expressions for lesser than (a comparison operator).
<=	Compares two expressions for lesser than or equal (a comparison operator).
>	Compares two expressions for greater than (a comparison operator).
>=	Compares two expressions for greater than or equal (a comparison operator).

OPERATOR	DESCRIPTION
<code>+</code>	Adds two numbers. This addition arithmetic operator can also add a number, in days, to a date.
<code>-</code>	Subtracts two numbers (an arithmetic subtraction operator).
<code>/</code>	Divides one number by another (an arithmetic division operator).
<code>*</code>	Multiplies two expressions (an arithmetic multiplication operator).
<code>%</code>	Returns the remainder of one number divided by another.
<code>AND</code>	Performs a bitwise logical AND operation between two integer values.
<code>OR</code>	Performs a bitwise logical OR operation between two specified integer values as translated to binary expressions within Transact-SQL statements.
<code>NOT</code>	Negates a Boolean input.
<code>CAST</code>	Converts an expression of one data type to another.
<code>BETWEEN</code>	Specifies a range to test.
<code>IN</code>	Determines whether a specified value matches any value in a subquery or a list.
<code>NULLIF</code>	Returns a null value if the two specified expressions are equal.
<code>COALESCE</code>	Evaluates the arguments in order and returns the current value of the first expression that initially doesn't evaluate to NULL.

If data types on the left and right of an operator are different, then automatic conversion will be performed according to the rules specified here: [Data type precedence \(Transact-SQL\)](#).

The query acceleration SQL language supports only a very small subset of the data types discussed in that article. See the [Data Types](#) section of this article.

## Casts

The query acceleration SQL language supports the CAST operator, according to the rules here: [Data type conversion \(Database Engine\)](#).

The query acceleration SQL language supports only a tiny subset of the data types discussed in that article. See the [Data Types](#) section of this article.

## String functions

The query acceleration SQL language supports the following standard SQL string functions:

FUNCTION	DESCRIPTION
CHAR_LENGTH	Returns the length in characters of the string expression, if the string expression is of a character data type; otherwise, returns the length in bytes of the string expression (the smallest integer not less than the number of bits divided by 8). (This function is the same as the CHARACTER_LENGTH function.)
CHARACTER_LENGTH	Returns the length in characters of the string expression, if the string expression is of a character data type; otherwise, returns the length in bytes of the string expression (the smallest integer not less than the number of bits divided by 8). (This function is the same as the CHAR_LENGTH function)
LOWER	Returns a character expression after converting uppercase character data to lowercase.
UPPER	Returns a character expression with lowercase character data converted to uppercase.
SUBSTRING	Returns part of a character, binary, text, or image expression in SQL Server.
TRIM	Removes the space character char(32) or other specified characters from the start and end of a string.
LEADING	Removes the space character char(32) or other specified characters from the start of a string.
TRAILING	Removes the space character char(32) or other specified characters from the end of a string.

Here's a few examples:

FUNCTION	EXAMPLE	RESULT
CHARACTER_LENGTH	<pre>SELECT CHARACTER_LENGTH('abcdefg') from BlobStorage</pre>	7
CHAR_LENGTH	<pre>SELECT CHAR_LENGTH(_1) from BlobStorage</pre>	1
LOWER	<pre>SELECT LOWER('AbCdEfG') from BlobStorage</pre>	abcdefg
UPPER	<pre>SELECT UPPER('AbCdEfG') from BlobStorage</pre>	ABCDEFG
SUBSTRING	<pre>SUBSTRING('123456789', 1, 5)</pre>	23456
TRIM	<pre>TRIM(BOTH '123' FROM '1112211Microsoft2221122')</pre>	Microsoft

## Date functions

The following standard SQL date functions are supported:

- `DATE_ADD`
- `DATE_DIFF`
- `EXTRACT`
- `TO_STRING`
- `TO_TIMESTAMP`

Currently, all [date formats of standard ISO8601](#) are converted.

#### **DATE\_ADD function**

The query acceleration SQL language supports year, month, day, hour, minute, second for the `DATE_ADD` function.

Examples:

```
DATE_ADD(datepart, quantity, timestamp)
DATE_ADD('minute', 1, CAST('2017-01-02T03:04:05.006Z' AS TIMESTAMP))
```

#### **DATE\_DIFF function**

The query acceleration SQL language supports year, month, day, hour, minute, second for the `DATE_DIFF` function.

```
DATE_DIFF(datepart, timestamp, timestamp)
DATE_DIFF('hour','2018-11-09T00:00+05:30','2018-11-09T01:00:23-08:00')
```

#### **EXTRACT function**

For EXTRACT other than date part supported for the `DATE_ADD` function, the query acceleration SQL language supports timezone\_hour and timezone\_minute as date part.

Examples:

```
EXTRACT(datepart FROM timestampstring)
EXTRACT(YEAR FROM '2010-01-01T')
```

#### **TO\_STRING function**

Examples:

```
TO_STRING(TimeStamp , format)
TO_STRING(CAST('1969-07-20T20:18Z' AS TIMESTAMP), 'MMMM d, y')
```

This table describes strings that you can use to specify the output format of the `TO_STRING` function.

FORMAT STRING	OUTPUT
yy	Year in 2 digit format - 1999 as '99'
y	Year in 4 digit format
yyyy	Year in 4 digit format
M	Month of year - 1
MM	Zero padded Month - 01

FORMAT STRING	OUTPUT
MMM	Abbr. month of Year - JAN
MMMM	Full month - May
d	Day of month (1-31)
dd	Zero padded day of Month (01-31)
a	AM or PM
h	Hour of day (1-12)
hh	Zero padded Hours od day (01-12)
H	Hour of day (0-23)
HH	Zero Padded hour of Day (00-23)
m	Minute of hour (0-59)
mm	Zero padded minute (00-59)
s	Second of Minutes (0-59)
ss	Zero padded Seconds (00-59)
S	Fraction of Seconds (0.1-0.9)
SS	Fraction of Seconds (0.01-0.99)
SSS	Fraction of Seconds (0.001-0.999)
X	Offset in Hours
XX or XXXX	Offset in hours and minutes (+0430)
XXX or XXXXX	Offset in hours and minutes (-07:00)
x	Offset in hours (7)
xx or xxxx	Offset in hour and minute (+0530)
Xxx or xxxx	Offset in hour and minute (+05:30)

#### TO\_TIMESTAMP function

Only ISO8601 formats are supported.

Examples:

```
TO_TIMESTAMP(string)
TO_TIMESTAMP('2007T')
```

#### NOTE

You can also use the `UTCNOW` function to get the system time.

## Aggregate Expressions

A SELECT statement may contain either one or more projection expressions or a single aggregate expression. The following aggregate expressions are supported:

EXPRESSION	DESCRIPTION
<code>COUNT(*)</code>	Returns the number of records which matched the predicate expression.
<code>COUNT(expression)</code>	Returns the number of records for which expression is non-null.
<code>AVG(expression)</code>	Returns the average of the non-null values of expression.
<code>MIN(expression)</code>	Returns the minimum non-null value of expression.
<code>MAX(expression)</code>	Returns the maximum non-null value of expression.
<code>SUM(expression)</code>	Returns the sum of all non-null values of expression.

## MISSING

The `IS MISSING` operator is the only non-standard that the query acceleration SQL language supports. For JSON data, if a field is missing from a particular input record, the expression field `IS MISSING` will evaluate to the Boolean value true.

## Table Descriptors

For CSV data, the table name is always `BlobStorage`. For example:

```
SELECT * FROM BlobStorage
```

For JSON data, additional options are available:

```
SELECT * FROM BlobStorage[*].path
```

This allows queries over subsets of the JSON data.

For JSON queries, you can mention the path in part of the FROM clause. These paths will help to parse the subset of JSON data. These paths can reference to JSON Array and Object values.

Let's take an example to understand this in more detail.

This is our sample data:

```
{
 "id": 1,
 "name": "mouse",
 "price": 12.5,
 "tags": [
 "wireless",
 "accessory"
],
 "dimensions": {
 "length": 3,
 "width": 2,
 "height": 2
 },
 "weight": 0.2,
 "warehouses": [
 {
 "latitude": 41.8,
 "longitude": -87.6
 }
]
}
```

You might be interested only in the `warehouses` JSON object from the above data. The `warehouses` object is a JSON array type, so you can mention this in the FROM clause. Your sample query can look something like this.

```
SELECT latitude FROM BlobStorage[*].warehouses[*]
```

The query gets all fields but selects only the latitude.

If you wanted to access only the `dimensions` JSON object value, you could use refer to that object in your query. For example:

```
SELECT length FROM BlobStorage[*].dimensions
```

This also limits your access to members of the `dimensions` object. If you want to access other members of JSON fields and inner values of JSON objects, then you might use a queries such as shown in the following example:

```
SELECT weight,warehouses[0].longitude,id,tags[1] FROM BlobStorage[*]
```

#### NOTE

`BlobStorage` and `BlobStorage[*]` both refer to the whole object. However, if you have a path in the `FROM` clause, then you'll need to use `BlobStorage[*].path`

## Sys.Split

This is a special form of the `SELECT` statement, which is available only for CSV-formatted data.

```
SELECT sys.split(split_size) FROM BlobStorage
```

Use this statement in cases where you want to download and then process CSV data records in batches. That way you can process records in parallel instead of having to download all records at one time. This statement doesn't return records from the CSV file. Instead, it returns a collection of batch sizes. You can then use each

batch size to retrieve a batch of data records.

Use the *split\_size* parameter to specify the number of bytes that you want each batch to contain. For example, if you want to process only 10 MB of data at a time, your statement would look like this:

`SELECT sys.split(10485760) FROM BlobStorage` because 10 MB is equal to 10,485,760 bytes. Each batch will contain as many records as can fit into those 10 MB.

In most cases, the size of each batch will be slightly higher than the number that you specify. That's because a batch cannot contain a partial record. If the last record in a batch starts before the end of your threshold, the batch will be larger so that it can contain the complete record. The size of the last batch will likely be smaller than the size that you specify.

**NOTE**

The *split\_size* must be at least 10 MB (10485760).

## See also

- [Azure Data Lake Storage query acceleration](#)
- [Filter data by using Azure Data Lake Storage query acceleration](#)

# azcopy

8/22/2022 • 2 minutes to read • [Edit Online](#)

Current version: 10.15.0

AzCopy is a command-line tool that moves data into and out of Azure Storage. See the [Get started with AzCopy](#) article to download AzCopy and learn about the ways that you can provide authorization credentials to the storage service.

## Synopsis

The general format of the commands is: `azcopy [command] [arguments] --[flag-name]=[flag-value]`.

To report issues or to learn more about the tool, see <https://github.com/Azure/azure-storage-azcopy>.

## Related conceptual articles

- [Get started with AzCopy](#)
- [Tutorial: Migrate on-premises data to cloud storage with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Options

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it is omitted, the throughput isn't capped.

`-h` , `--help` help for azcopy

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is `'.core.windows.net;core.chinacloudapi.cn;core.cloudapi.de;core.usgovcloudapi.net;*.storage.azure.net'`. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [Get started with AzCopy](#)
- [azcopy bench](#)
- [azcopy copy](#)
- [azcopy doc](#)
- [azcopy env](#)
- [azcopy jobs](#)
- [azcopy jobs clean](#)
- [azcopy jobs list](#)
- [azcopy jobs remove](#)
- [azcopy jobs resume](#)

- [azcopy jobs show](#)
- [azcopy list](#)
- [azcopy login](#)
- [azcopy login status](#)
- [azcopy logout](#)
- [azcopy make](#)
- [azcopy remove](#)
- [azcopy sync](#)
- [azcopy set-properties](#)

# azcopy bench

8/22/2022 • 4 minutes to read • [Edit Online](#)

Runs a performance benchmark by uploading or downloading test data to or from a specified destination. For uploads, the test data is automatically generated.

The benchmark command runs the same process as 'copy', except that:

- Instead of requiring both source and destination parameters, benchmark takes just one. This is the blob container, Azure Files Share, or Azure Data Lake Storage Gen2 file system that you want to upload to or download from.
- The 'mode' parameter describes whether AzCopy should test uploads to or downloads from given target. Valid values are 'Upload' and 'Download'. Default value is 'Upload'.
- For upload benchmarks, the payload is described by command line parameters, which control how many files are auto-generated and how big they are. The generation process takes place entirely in memory. Disk isn't used.
- For downloads, the payload consists of whichever files already exist at the source. (See example below about how to generate test files if needed).
- Only a few of the optional parameters that are available to the copy command are supported.
- Additional diagnostics are measured and reported.
- For uploads, the default behavior is to delete the transferred data at the end of the test run. For downloads, the data is never actually saved locally.

Benchmark mode will automatically tune itself to the number of parallel TCP connections that gives the maximum throughput. It will display that number at the end. To prevent auto-tuning, set the COPY\_CONCURRENCY\_VALUE environment variable to a specific number of connections.

All the usual authentication types are supported. However, the most convenient approach for benchmarking upload is typically to create an empty container with a SAS token and use SAS authentication. (Download mode requires a set of test data to be present in the target container.)

```
azcopy bench [destination] [flags]
```

## Examples

Run an upload benchmark with default parameters (suitable for benchmarking networks up to 1 Gbps):

```
azcopy bench "https://[account].blob.core.windows.net/[container]?<SAS>"
```

Run a benchmark test that uploads 100 files, each 2 GiB in size: (suitable for benchmarking on a fast network, e.g. 10 Gbps):

```
azcopy bench "https://[account].blob.core.windows.net/[container]?<SAS>" --file-count 100 --size-per-file 2G
```

Same as above, but use 50,000 files, each 8 MiB in size and compute their MD5 hashes (in the same way that the --put-md5 flag does this in the copy command). The purpose of --put-md5 when benchmarking is to test whether MD5 computation affects throughput for the selected file count and size:

```
azcopy bench --mode='Upload' "https://[account].blob.core.windows.net/[container]?<SAS>" --file-count 50000 -
-size-per-file 8M --put-md5
```

Run a benchmark test that downloads existing files from a target

```
azcopy bench --mode='Download' "https://[account].blob.core.windows.net/[container]?<SAS?!"
```

Run an upload that doesn't delete the transferred files. (These files can then serve as the payload for a download test)

```
azcopy bench "https://[account].blob.core.windows.net/[container]?<SAS>" --file-count 100 --delete-test-data=false
```

## Options

`--blob-type string` defines the type of blob at the destination. Used to allow benchmarking different blob types. Identical to the same-named parameter in the copy command (default "Detect")

`--block-size-mb float` Use this block size (specified in MiB). Default is automatically calculated based on file size. Decimal fractions are allowed - for example, 0.25. Identical to the same-named parameter in the copy command

`--check-length` Check the length of a file on the destination after the transfer. If there's a mismatch between source and destination, the transfer is marked as failed. (default true)

`--delete-test-data` If true, the benchmark data will be deleted at the end of the benchmark run. Set it to false if you want to keep the data at the destination - for example, to use it for manual tests outside benchmark mode (default true)

`--file-count` (uint) number of auto-generated data files to use (default 100)

`-h , --help` help for bench

`--log-level` (string) define the log verbosity for the log file, available levels: INFO(all requests/responses), WARNING(slow responses), ERROR(only failed requests), and NONE(no output logs). (default "INFO")

`--mode` (string) Defines if Azcopy should test uploads or downloads from this target. Valid values are 'upload' and 'download'. Defaulted option is 'upload'. (default "upload")

`--number-of-folders` (uint) If larger than 0, create folders to divide up the data.

`--put-md5` Create an MD5 hash of each file, and save the hash as the Content-MD5 property of the destination blob/file. (By default the hash is NOT created.) Identical to the same-named parameter in the copy command

`--size-per-file` (string) Size of each auto-generated data file. Must be a number immediately followed by K, M or G. E.g. 12k or 200G (default "250M")

## Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it's omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)

# azcopy copy

8/22/2022 • 15 minutes to read • [Edit Online](#)

Copies source data to a destination location.

## Synopsis

Copies source data to a destination location. The supported directions are:

- local <-> Azure Blob (SAS or OAuth authentication)
- local <-> Azure Files (Share/directory SAS authentication)
- local <-> Azure Data Lake Storage Gen2 (SAS, OAuth, or SharedKey authentication)
- Azure Blob (SAS or public) -> Azure Blob (SAS or OAuth authentication)
- Azure Blob (SAS or public) -> Azure Files (SAS)
- Azure Files (SAS) -> Azure Files (SAS)
- Azure Files (SAS) -> Azure Blob (SAS or OAuth authentication)
- AWS S3 (Access Key) -> Azure Block Blob (SAS or OAuth authentication)
- Google Cloud Storage (Service Account Key) -> Azure Block Blob (SAS or OAuth authentication)

Refer to the examples for more information.

## Advanced

AzCopy automatically detects the content type of the files when uploading from the local disk, based on the file extension or content (if no extension is specified).

The built-in lookup table is small, but on Unix, it's augmented by the local system's mime.types file(s) if available under one or more of these names:

- /etc/mime.types
- /etc/apache2/mime.types
- /etc/apache/mime.types

On Windows, MIME types are extracted from the registry. This feature can be turned off with the help of a flag. Refer to the flag section.

If you set an environment variable by using the command line, that variable will be readable in your command line history. Consider clearing variables that contain credentials from your command line history. To keep variables from appearing in your history, you can use a script to prompt the user for their credentials, and to set the environment variable.

```
azcopy copy [source] [destination] [flags]
```

## Examples

Upload a single file by using OAuth authentication. If you haven't yet logged into AzCopy, please run the azcopy login command before you run the following command.

```
azcopy cp "/path/to/file.txt" "https://[account].blob.core.windows.net/[container]/[path/to/blob]"
```

Same as above, but this time also compute MD5 hash of the file content and save it as the blob's Content-MD5

property:

```
azcopy cp "/path/to/file.txt" "https://[account].blob.core.windows.net/[container]/[path/to/blob]" --put-md5
```

Upload a single file by using a SAS token:

```
azcopy cp "/path/to/file.txt" "https://[account].blob.core.windows.net/[container]/[path/to/blob]?[SAS]"
```

Upload a single file by using a SAS token and piping (block blobs only):

```
cat "/path/to/file.txt" | azcopy cp "https://[account].blob.core.windows.net/[container]/[path/to/blob]?[SAS]" --from-to PipeBlob
```

Upload a single file by using OAuth and piping (block blobs only):

```
cat "/path/to/file.txt" | azcopy cp "https://[account].blob.core.windows.net/[container]/[path/to/blob]" --from-to PipeBlob
```

Upload an entire directory by using a SAS token:

```
azcopy cp "/path/to/dir" "https://[account].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" --recursive=true
```

or

```
azcopy cp "/path/to/dir" "https://[account].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" --recursive=true --put-md5
```

Upload a set of files by using a SAS token and wildcard (\*) characters:

```
azcopy cp "/path/*foo/*bar/*.pdf" "https://[account].blob.core.windows.net/[container]/[path/to/directory]?[SAS]"
```

Upload files and directories by using a SAS token and wildcard (\*) characters:

```
azcopy cp "/path/*foo/*bar*" "https://[account].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" --recursive=true
```

Upload files and directories to Azure Storage account and set the query-string encoded tags on the blob.

- To set tags {key = "bla bla", val = "foo"} and {key = "bla bla 2", val = "bar"}, use the following syntax:

```
azcopy cp "/path/*foo/*bar*" "https://[account].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" --blob-tags="bla%20bla=foo&bla%20bla%202=bar"
```

- Keys and values are URL encoded and the key-value pairs are separated by an ampersand(&)
- While setting tags on the blobs, there are additional permissions('t' for tags) in SAS without which the service will give authorization error back.

Download a single file by using OAuth authentication. If you haven't yet logged into AzCopy, please run the azcopy login command before you run the following command.

```
azcopy cp "https://[account].blob.core.windows.net/[container]/[path/to/blob]" "/path/to/file.txt"
```

Download a single file by using a SAS token:

```
azcopy cp "https://[account].blob.core.windows.net/[container]/[path/to/blob]?[SAS]" "/path/to/file.txt"
```

Download a single file by using a SAS token and then piping the output to a file (block blobs only):

```
azcopy cp "https://[account].blob.core.windows.net/[container]/[path/to/blob]?[SAS]" --from-to BlobPipe > "/path/to/file.txt"
```

Download a single file by using OAuth and then piping the output to a file (block blobs only):

```
azcopy cp "https://[account].blob.core.windows.net/[container]/[path/to/blob]" --from-to BlobPipe > "/path/to/file.txt"
```

Download an entire directory by using a SAS token:

```
azcopy cp "https://[account].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" "/path/to/dir" --recursive=true
```

A note about using a wildcard character (\*) in URLs:

There's only two supported ways to use a wildcard character in a URL.

- You can use one just after the final forward slash (/) of a URL. This copies all of the files in a directory directly to the destination without placing them into a subdirectory.
- You can also use one in the name of a container as long as the URL refers only to a container and not to a blob. You can use this approach to obtain files from a subset of containers.

Download the contents of a directory without copying the containing directory itself.

```
azcopy cp "https://[srcaccount].blob.core.windows.net/[container]/[path/to/folder]/*?[SAS]" "/path/to/dir"
```

Download an entire storage account.

```
azcopy cp "https://[srcaccount].blob.core.windows.net/" "/path/to/dir" --recursive
```

Download a subset of containers within a storage account by using a wildcard symbol (\*) in the container name.

```
azcopy cp "https://[srcaccount].blob.core.windows.net/[container*name]" "/path/to/dir" --recursive
```

Download all the versions of a blob from Azure Storage to local directory. Ensure that source is a valid blob, destination is a local folder and `versionidsFile` which takes in a path to the file where each version is written on a separate line. All the specified versions will get downloaded in the destination folder specified.

```
azcopy cp "https://[srcaccount].blob.core.windows.net/[containername]/[blobname]" "/path/to/dir" --list-of-versions="/another/path/to/dir/[versionidsFile]"
```

Copy a single blob to another blob by using a SAS token.

```
azcopy cp "https://[srcaccount].blob.core.windows.net/[container]/[path/to/blob]?[SAS]" "https://[destaccount].blob.core.windows.net/[container]/[path/to/blob]?[SAS]"
```

Copy a single blob to another blob by using a SAS token and an OAuth token.

```
azcopy cp "https://[srcaccount].blob.core.windows.net/[container]/[path/to/blob]" "https://[destaccount].blob.core.windows.net/[container]/[path/to/blob]"
```

Copy one blob virtual directory to another by using a SAS token:

```
azcopy cp "https://[srcaccount].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" "https://[destaccount].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" --recursive=true
```

Copy all blob containers, directories, and blobs from storage account to another by using a SAS token:

```
azcopy cp "https://[srcaccount].blob.core.windows.net?[SAS]" "https://[destaccount].blob.core.windows.net?[SAS]" --recursive=true
```

Copy a single object to Blob Storage from Amazon Web Services (AWS) S3 by using an access key and a SAS token. First, set the environment variable AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY for AWS S3 source.

```
azcopy cp "https://s3.amazonaws.com/[bucket]/[object]" "https://[destaccount].blob.core.windows.net/[container]/[path/to/blob]?[SAS]"
```

Copy an entire directory to Blob Storage from AWS S3 by using an access key and a SAS token. First, set the environment variable AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY for AWS S3 source.

```
azcopy cp "https://s3.amazonaws.com/[bucket]/[folder]" "https://[destaccount].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" --recursive=true
```

Refer to <https://docs.aws.amazon.com/AmazonS3/latest/user-guide/using-folders.html> to better understand the

[folder] placeholder.

Copy all buckets to Blob Storage from Amazon Web Services (AWS) by using an access key and a SAS token. First, set the environment variable AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY for AWS S3 source.

```
azcopy cp "https://s3.amazonaws.com/" "https://[destaccount].blob.core.windows.net?[SAS]" --recursive=true
```

Copy all buckets to Blob Storage from an Amazon Web Services (AWS) region by using an access key and a SAS token. First, set the environment variable AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY for AWS S3 source.

```
azcopy cp "https://s3-[region].amazonaws.com/" "https://[destaccount].blob.core.windows.net?[SAS]" --recursive=true
```

Copy a subset of buckets by using a wildcard symbol (\*) in the bucket name. Like the previous examples, you'll need an access key and a SAS token. Make sure to set the environment variable AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY for AWS S3 source.

```
azcopy cp "https://s3.amazonaws.com/[bucket*name]/" "https://[destaccount].blob.core.windows.net?[SAS]" --recursive=true
```

Copy blobs from one blob storage to another and preserve the tags from source. To preserve tags, use the following syntax:

```
azcopy cp "https://[account].blob.core.windows.net/[source_container]/[path/to/directory]?[SAS]"
"https://[account].blob.core.windows.net/[destination_container]/[path/to/directory]?[SAS]" --s2s-preserve-blob-tags=true
```

Transfer files and directories to Azure Storage account and set the given query-string encoded tags on the blob.

- To set tags {key = "bla bla", val = "foo"} and {key = "bla bla 2", val = "bar"}, use the following syntax:

```
azcopy cp "https://[account].blob.core.windows.net/[source_container]/[path/to/directory]?[SAS]"
"https://[account].blob.core.windows.net/[destination_container]/[path/to/directory]?[SAS]" --blob-tags="bla%20bla=foo&bla%20bla%202=bar"
```

- Keys and values are URL encoded and the key-value pairs are separated by an ampersand(&)
- While setting tags on the blobs, there are additional permissions('t' for tags) in SAS without which the service will give authorization error back.

Copy a single object to Blob Storage from Google Cloud Storage (GCS) by using a service account key and a SAS token. First, set the environment variable GOOGLE\_APPLICATION\_CREDENTIALS for GCS source.

```
azcopy cp "https://storage.cloud.google.com/[bucket]/[object]"
"https://[destaccount].blob.core.windows.net/[container]/[path/to/blob]?[SAS]"
```

Copy an entire directory to Blob Storage from Google Cloud Storage (GCS) by using a service account key and a SAS token. First, set the environment variable GOOGLE\_APPLICATION\_CREDENTIALS for GCS source.

```
azcopy cp "https://storage.cloud.google.com/[bucket]/[folder]"
"https://[destaccount].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" --recursive=true
```

Copy an entire bucket to Blob Storage from Google Cloud Storage (GCS) by using a service account key and a SAS token. First, set the environment variable GOOGLE\_APPLICATION\_CREDENTIALS for GCS source.

```
azcopy cp "https://storage.cloud.google.com/[bucket]" "https://[destaccount].blob.core.windows.net/?[SAS]" --recursive=true
```

Copy all buckets to Blob Storage from Google Cloud Storage (GCS) by using a service account key and a SAS token. First, set the environment variables GOOGLE\_APPLICATION\_CREDENTIALS and GOOGLE\_CLOUD\_PROJECT=<project-id> for GCS source

```
azcopy cp "https://storage.cloud.google.com/" "https://[destaccount].blob.core.windows.net/?[SAS]" --recursive=true
```

Copy a subset of buckets by using a wildcard symbol (\*) in the bucket name from Google Cloud Storage (GCS) by using a service account key and a SAS token for destination. First, set the environment variables

```
GOOGLE_APPLICATION_CREDENTIALS and GOOGLE_CLOUD_PROJECT=<project-id> for GCS source
```

```
azcopy cp "https://storage.cloud.google.com/[bucket*name]/" "https://[destaccount].blob.core.windows.net/?[SAS]" --recursive=true
```

## Options

**--as-subdir** True by default. Places folder sources as subdirectories under the destination. (default true)

**--backup** Activates Windows' SeBackupPrivilege for uploads, or SeRestorePrivilege for downloads, to allow AzCopy to see read all files, regardless of their file system permissions, and to restore all permissions. Requires that the account running AzCopy already has these permissions (for example, has Administrator rights or is a member of the 'Backup Operators' group). This flag activates privileges that the account already has

**--blob-tags** (string) Set tags on blobs to categorize data in your storage account

**--blob-type** (string) Defines the type of blob at the destination. This is used for uploading blobs and when copying between accounts (default 'Detect'). Valid values include 'Detect', 'BlockBlob', 'PageBlob', and 'AppendBlob'. When copying between accounts, a value of 'Detect' causes AzCopy to use the type of source blob to determine the type of the destination blob. When uploading a file, 'Detect' determines if the file is a VHD or a VHDX file based on the file extension. If the file is either a VHD or VHDX file, AzCopy treats the file as a page blob. (default "Detect")

**--block-blob-tier** (string) upload block blob to Azure Storage using this blob tier. (default "None")

**--block-size-mb** (float) Use this block size (specified in MiB) when uploading to Azure Storage, and downloading from Azure Storage. The default value is automatically calculated based on file size. Decimal fractions are allowed (For example: 0.25).

**--cache-control** (string) Set the cache-control header. Returned on download.

**--check-length** Check the length of a file on the destination after the transfer. If there's a mismatch between source and destination, the transfer is marked as failed. (default true)

**--check-md5** (string) Specifies how strictly MD5 hashes should be validated when downloading. Only available when downloading. Available options: NoCheck, LogOnly, FailIfDifferent, FailIfDifferentOrMissing. (default 'FailIfDifferent') (default "FailIfDifferent")

**--content-disposition** (string) Set the content-disposition header. Returned on download.

**--content-encoding** (string) Set the content-encoding header. Returned on download.

**--content-language** (string) Set the content-language header. Returned on download.

**--content-type** (string) Specifies the content type of the file. Implies no-guess-mime-type. Returned on download.

**--cpk-by-name** (string) Client provided key by name that lets clients making requests against Azure Blob storage an option to provide an encryption key on a per-request basis. Provided key name will be fetched from Azure Key Vault and will be used to encrypt the data

**--cpk-by-value** Client provided key by name that let clients making requests against Azure Blob storage an option to provide an encryption key on a per-request basis. Provided key and its hash will be fetched from environment variables

**--decompress** Automatically decompress files when downloading, if their content-encoding indicates that they're compressed. The supported content-encoding values are 'gzip' and 'deflate'. File extensions of '.gz'/.gzip'

or '.zz' aren't necessary, but will be removed if present.

**--disable-auto-decoding** False by default to enable automatic decoding of illegal chars on Windows. Can be set to true to disable automatic decoding.

**--dry-run** Prints the file paths that would be copied by this command. This flag doesn't copy the actual files.

**--exclude-attributes** (string) (Windows only) Exclude files whose attributes match the attribute list. For example: A;S;R

**--exclude-blob-type** (string) Optionally specifies the type of blob (BlockBlob/ PageBlob/ AppendBlob) to exclude when copying blobs from the container or the account. Use of this flag isn't applicable for copying data from non azure-service to service. More than one blob should be separated by ':'.

**--exclude-path** (string) Exclude these paths when copying. This option doesn't support wildcard characters (\*). Checks relative path prefix(For example: myFolder;myFolder/subDirName/file.pdf). When used in combination with account traversal, paths don't include the container name.

**--exclude-pattern** (string) Exclude these files when copying. This option supports wildcard characters (\*)

**--exclude-regex** (string) Exclude all the relative path of the files that align with regular expressions. Separate regular expressions with ':'.

**--follow-symlinks** Follow symbolic links when uploading from local file system.

**--force-if-read-only** When overwriting an existing file on Windows or Azure Files, force the overwrite to work even if the existing file has its read-only attribute set

**--from-to** (string) Optionally specifies the source destination combination. For Example: LocalBlob, BlobLocal, LocalBlobFS. Piping: BlobPipe, PipeBlob

**-h , --help** help for copy

**--include-after** (string) Include only those files modified on or after the given date/time. The value should be in ISO8601 format. If no timezone is specified, the value is assumed to be in the local timezone of the machine running AzCopy. E.g., `2020-08-19T15:04:00Z` for a UTC time, or `2020-08-19` for midnight (00:00) in the local timezone. As of AzCopy 10.5, this flag applies only to files, not folders, so folder properties won't be copied when using this flag with `--preserve-smb-info` or `--preserve-smb-permissions`.

**--include-attributes** (string) (Windows only) Include files whose attributes match the attribute list. For example: A;S;R

**--include-before** (string) Include only those files modified before or on the given date/time. The value should be in ISO8601 format. If no timezone is specified, the value is assumed to be in the local timezone of the machine running AzCopy. for example, `2020-08-19T15:04:00Z` for a UTC time, or `2020-08-19` for midnight (00:00) in the local timezone. As of AzCopy 10.7, this flag applies only to files, not folders, so folder properties won't be copied when using this flag with `--preserve-smb-info` or `--preserve-smb-permissions`.

**--include-directory-stub** False by default to ignore directory stubs. Directory stubs are blobs with metadata `hdi_isfolder:true`. Setting value to true will preserve directory stubs during transfers.

**--include-path** (string) Include only these paths when copying. This option doesn't support wildcard characters (\*). Checks relative path prefix (For example: myFolder;myFolder/subDirName/file.pdf).

**--include-pattern** (string) Include only these files when copying. This option supports wildcard characters (\*). Separate files by using a ':'.

**--include-regex** (string) Include only the relative path of the files that align with regular expressions. Separate regular expressions with ':'.

**--list-of-versions** (string) Specifies a file where each version ID is listed on a separate line. Ensure that the source must point to a single blob and all the version IDs specified in the file using this flag must belong to the source blob only. AzCopy will download the specified versions in the destination folder provided.

**--log-level** (string) Define the log verbosity for the log file, available levels: INFO(all requests/responses), WARNING(slow responses), ERROR(only failed requests), and NONE(no output logs). (default 'INFO'). (default "INFO")

**--metadata** (string) Upload to Azure Storage with these key-value pairs as metadata.

**--no-guess-mime-type** Prevents AzCopy from detecting the content-type based on the extension or content of the file.

**--overwrite** (string) Overwrite the conflicting files and blobs at the destination if this flag is set to true. (default 'true') Possible values include 'true', 'false', 'prompt', and 'ifSourceNewer'. For destinations that support folders, conflicting folder-level properties will be overwritten this flag is 'true' or if a positive response is provided to the prompt. (default "true")

**--page-blob-tier** (string) Upload page blob to Azure Storage using this blob tier. (default 'None'). (default "None")

**--preserve-last-modified-time** Only available when destination is file system.

**--preserve-owner** Only has an effect in downloads, and only when **--preserve-smb-permissions** is used. If true (the default), the file Owner and Group are preserved in downloads. If set to false,

**--preserve-smb-permissions** will still preserve ACLs but Owner and Group will be based on the user running AzCopy (default true)

**--preserve-permissions** False by default. Preserves ACLs between aware resources (Windows and Azure Files, or Azure Data Lake Storage Gen2 to Azure Data Lake Storage Gen2). For Hierarchical Namespace accounts, you'll need a container SAS or OAuth token with Modify Ownership and Modify Permissions permissions. For downloads, you'll also need the **--backup** flag to restore permissions where the new Owner won't be the user running AzCopy. This flag applies to both files and folders, unless a file-only filter is specified (for example, include-pattern).

**--preserve-smb-info** For SMB-aware locations, flag will be set to true by default. Preserves SMB property info (last write time, creation time, attribute bits) between SMB-aware resources (Windows and Azure Files). Only the attribute bits supported by Azure Files will be transferred; any others will be ignored. This flag applies to both files and folders, unless a file-only filter is specified (for example, include-pattern). The info transferred for folders is the same as that for files, except for **Last Write Time** which is never preserved for folders. (default true)

**--put-md5** Create an MD5 hash of each file, and save the hash as the Content-MD5 property of the destination blob or file. (By default the hash is NOT created.) Only available when uploading.

**--recursive** Look into subdirectories recursively when uploading from local file system.

**--s2s-detect-source-changed** Detect if the source file/blob changes while it is being read. (This parameter only applies to service-to-service copies, because the corresponding check is permanently enabled for uploads and downloads.)

**--s2s-handle-invalid-metadata** (string) Specifies how invalid metadata keys are handled. Available options: ExcludelfInvalid, FaillfInvalid, RenamelfInvalid. (default 'ExcludelfInvalid'). (default "ExcludelfInvalid")

**--s2s-preserve-access-tier** Preserve access tier during service to service copy. Refer to [Azure Blob storage: hot, cool, and archive access tiers](#) to ensure destination storage account supports setting access tier. In the cases that setting access tier isn't supported, make sure to use s2sPreserveAccessTier=false to bypass copying access tier.

(default true). (default true)

`--s2s-preserve-blob-tags` Preserve index tags during service to service transfer from one blob storage to another

`--s2s-preserve-properties` Preserve full properties during service to service copy. For AWS S3 and Azure File non-single file source, the list operation doesn't return full properties of objects and files. To preserve full properties, AzCopy needs to send one more request per object or file. (default true)

## Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it's omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)

# azcopy doc

8/22/2022 • 2 minutes to read • [Edit Online](#)

Generates documentation for the tool in Markdown format.

## Synopsis

Generates documentation for the tool in Markdown format, and stores them in the designated location.

By default, the files are stored in a folder named 'doc' inside the current directory.

```
azcopy doc [flags]
```

## Options

`-h` , `--help` help for doc `--output-location` (string) where to put the generated markdown files (default "./doc")

## Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it's omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text").

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;\*.core.cloudapi.

## See also

- [azcopy](#)

# azcopy env

8/22/2022 • 2 minutes to read • [Edit Online](#)

Shows the environment variables that can configure AzCopy's behavior. For a complete list of environment variables, see [AzCopy v10 configuration settings \(Azure Storage\)](#).

## Synopsis

Shows the environment variables that you can use to configure the behavior of AzCopy.

If you set an environment variable by using the command line, that variable will be readable in your command line history. Consider clearing variables that contain credentials from your command line history. To keep variables from appearing in your history, you can use a script to prompt the user for their credentials, and to set the environment variable.

```
azcopy env [flags]
```

## Options

`-h`, `--help` help for env `--show-sensitive` Shows sensitive/secret environment variables.

## Options inherited from parent commands

`--cap-mbps float` Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it's omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;\*.core.cloudapi.de;.core.usgovcloudapi.net;storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)

# azcopy jobs

8/22/2022 • 2 minutes to read • [Edit Online](#)

Subcommands related to managing jobs.

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Examples

```
azcopy jobs show [jobID]
```

## Options

`-h` , `--help` Help for jobs

## Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it is omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;.core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)
- [azcopy jobs list](#)
- [azcopy jobs resume](#)
- [azcopy jobs show](#)

# azcopy jobs clean

8/22/2022 • 2 minutes to read • [Edit Online](#)

Remove all log and plan files for all jobs

```
azcopy jobs clean [flags]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Examples

```
azcopy jobs clean --with-status=completed
```

## Options

`-h` , `--help` help for clean `--with-status` (string) only remove the jobs with this status, available values: All, Canceled, Failed, Completed, CompletedWithErrors, CompletedWithSkipped, CompletedWithErrorsAndSkipped (default "All")

## Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it is omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;.core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy jobs](#)

# azcopy jobs list

8/22/2022 • 2 minutes to read • [Edit Online](#)

Displays information on all jobs.

## Synopsis

```
azcopy jobs list [flags]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Options

`-h` , `--help` help for list `--with-status` (string) List the jobs with given status, available values: All, Canceled, Failed, InProgress, Completed, CompletedWithErrors, CompletedWithFailures, CompletedWithErrorsAndSkipped (default "All")

## Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it is omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;.core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy jobs](#)

# azcopy jobs remove

8/22/2022 • 2 minutes to read • [Edit Online](#)

Remove all files associated with the given job ID.

## NOTE

You can customize the location where log and plan files are saved. See the [azcopy env](#) command to learn more.

```
azcopy jobs remove [jobID] [flags]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Examples

```
azcopy jobs rm e52247de-0323-b14d-4cc8-76e0be2e2d44
```

## Options

`--help` Help for remove.

## Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it's omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;.core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy jobs](#)

# azcopy jobs resume

8/22/2022 • 2 minutes to read • [Edit Online](#)

Resumes the existing job with the given job ID.

## Synopsis

```
azcopy jobs resume [jobID] [flags]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Options

`--destination-sas` (string) destination SAS token of the destination for a given Job ID.

`--exclude` (string) Filter: exclude these failed transfer(s) when resuming the job. Files should be separated by ':'.

`-h` , `--help` help for resume

`--include` (string) Filter: only include these failed transfer(s) when resuming the job. Files should be separated by ':'.

`--source-sas` (string) Source SAS token of the source for a given Job ID.

## Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it is omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is

'.core.windows.net;core.chinacloudapi.cn;.core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy jobs](#)

# azcopy jobs show

8/22/2022 • 2 minutes to read • [Edit Online](#)

Shows detailed information for the given job ID.

## Synopsis

If only the job ID is supplied without a flag, then the progress summary of the job is returned.

The byte counts and percent complete that appears when you run this command reflect only files that are completed in the job. They don't reflect partially completed files.

If the `--with-status` flag is set, then the list of transfers in the job with the given value will be shown.

```
azcopy jobs show [jobID] [flags]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Options

`-h` , `--help` Help for show `--with-status` (string) Only list the transfers of job with this status, available values: Started, Success, Failed.

## Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it is omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;.core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy jobs](#)

# azcopy list

8/22/2022 • 2 minutes to read • [Edit Online](#)

Lists the entities in a given resource.

## Synopsis

Only Blob containers are supported in the current release.

```
azcopy list [containerURL] [flags]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Examples

```
azcopy list [containerURL] --properties [semicolon(;) separated list of attributes (LastModifiedTime, VersionId, BlobType, BlobAccessTier, ContentType, ContentEncoding, LeaseState, LeaseDuration, LeaseStatus) enclosed in double quotes ("")]
```

## Options

`-h` , `--help` Help for list

`--machine-readable` Lists file sizes in bytes.

`--mega-units` Displays units in orders of 1000, not 1024.

`--properties` (string) delimiter (:) separated values of properties required in list output.

`--running-tally` Counts the total number of files and their sizes.

## Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it's omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;.core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)

# azcopy login

8/22/2022 • 3 minutes to read • [Edit Online](#)

Logs in to Azure Active Directory to access Azure Storage resources.

## Synopsis

Log in to Azure Active Directory to access Azure Storage resources.

To be authorized to your Azure Storage account, you must assign the **Storage Blob Data Contributor** role to your user account in the context of either the Storage account, parent resource group, or parent subscription.

This command will cache encrypted login information for current user using the OS built-in mechanisms.

### IMPORTANT

If you set an environment variable by using the command line, that variable will be readable in your command line history. Consider clearing variables that contain credentials from your command line history. To keep variables from appearing in your history, you can use a script to prompt the user for their credentials, and to set the environment variable.

```
azcopy login [flags]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Tutorial: Migrate on-premises data to cloud storage with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Examples

Log in interactively with default AAD tenant ID set to common:

```
azcopy login
```

Log in interactively with a specified tenant ID:

```
azcopy login --tenant-id "[TenantID]"
```

Log in by using the system-assigned identity of a Virtual Machine (VM):

```
azcopy login --identity
```

Log in by using the user-assigned identity of a VM and a Client ID of the service identity:

```
azcopy login --identity --identity-client-id "[ServiceIdentityClientID]"
```

Log in by using the user-assigned identity of a VM and an Object ID of the service identity:

```
azcopy login --identity --identity-object-id "[ServiceIdentityObjectID]"
```

Log in by using the user-assigned identity of a VM and a Resource ID of the service identity:

```
azcopy login --identity --identity-resource-id
"/subscriptions/<subscriptionId>/resourcegroups/myRG/providers/Microsoft.ManagedIdentity/userAssignedIdentities/myID"
```

Log in as a service principal by using a client secret:

Set the environment variable AZCOPY\_SPA\_CLIENT\_SECRET to the client secret for secret based service principal auth.

```
azcopy login --service-principal --application-id <your service principal's application ID>
```

Log in as a service principal by using a certificate and it's password:

Set the environment variable AZCOPY\_SPA\_CERT\_PASSWORD to the certificate's password for cert based service principal auth

```
azcopy login --service-principal --certificate-path /path/to/my/cert --application-id <your service principal's application ID>
```

Treat /path/to/my/cert as a path to a PEM or PKCS12 file--. AzCopy doesn't reach into the system cert store to obtain your certificate. --certificate-path is mandatory when doing cert-based service principal auth.

Subcommand for login to check the login status of your current session.

```
azcopy login status
```

## Options

--aad-endpoint (string) The Azure Active Directory endpoint to use. The default (<https://login.microsoftonline.com>) is correct for the global Azure cloud. Set this parameter when authenticating in a national cloud. Not needed for Managed Service Identity

--application-id (string) Application ID of user-assigned identity. Required for service principal auth.

--certificate-path (string) Path to certificate for SPN authentication. Required for certificate-based service principal auth.

-h , --help Help for login

--identity Log in using virtual machine's identity, also known as managed service identity (MSI).

--identity-client-id (string) Client ID of user-assigned identity.

--identity-object-id (string) Object ID of user-assigned identity.

--identity-resource-id (string) Resource ID of user-assigned identity.

--service-principal Log in via Service Principal Name (SPN) by using a certificate or a secret. The client secret or certificate password must be placed in the appropriate environment variable. Type AzCopy env to see names and descriptions of environment variables.

--tenant-id (string) The Azure Active Directory tenant ID to use for OAuth device interactive login.

## Options inherited from parent commands

--cap-mbps (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it's omitted, the throughput isn't capped.

--output-type (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

--trusted-microsoft-suffixes (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)

# azcopy logout

8/22/2022 • 2 minutes to read • [Edit Online](#)

Logs the user out and terminates access to Azure Storage resources.

## Synopsis

This command will remove all the cached login information for the current user.

```
azcopy logout [flags]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Options

`-h` , `--help` help for logout

### Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it is omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is

'`.core.windows.net;core.chinacloudapi.cn;core.cloudapi.de;core.usgovcloudapi.net;*.storage.azure.net`'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)

# azcopy make

8/22/2022 • 2 minutes to read • [Edit Online](#)

Creates a container or file share.

## Synopsis

Create a container or file share represented by the given resource URL.

```
azcopy make [resourceURL] [flags]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Examples

```
azcopy make "https://[account-name].[blob,file,dfs].core.windows.net/[top-level-resource-name]"
```

## Options

`-h` , `--help` help for make `--quota-gb` (uint32) Specifies the maximum size of the share in gigabytes (GiB), 0 means you accept the file service's default quota.

## Options inherited from parent commands

`--cap-mbps` (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it is omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;.core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)

# azcopy remove

8/22/2022 • 3 minutes to read • [Edit Online](#)

Delete blobs or files from an Azure storage account.

## Synopsis

```
azcopy remove [resourceURL] [flags]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)

## Examples

Remove a single blob by using a SAS token:

```
azcopy rm "https://[account].blob.core.windows.net/[container]/[path/to/blob]?[SAS]"
```

Remove an entire virtual directory by using a SAS token:

```
azcopy rm "https://[account].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" --recursive=true
```

Remove only the blobs inside of a virtual directory, but don't remove any subdirectories or blobs within those subdirectories:

```
azcopy rm "https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]" --recursive=false
```

Remove a subset of blobs in a virtual directory (For example: remove only jpg and pdf files, or if the blob name is "exactName"):

```
azcopy rm "https://[account].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" --recursive=true --include-pattern="*.jpg;*.pdf;exactName"
```

Remove an entire virtual directory but exclude certain blobs from the scope (For example: every blob that starts with foo or ends with bar):

```
azcopy rm "https://[account].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" --recursive=true --exclude-pattern="foo*;*bar"
```

Remove specified version IDs of a blob from Azure Storage. Ensure that source is a valid blob and `versionidsfile` which takes in a path to the file where each version is written on a separate line. All the specified versions will be removed from Azure Storage.

```
azcopy rm "https://[srcaccount].blob.core.windows.net/[containername]/[blobname]" "/path/to/dir" --list-of-versions="/path/to/dir/[versionidsfile]"
```

Remove specific blobs and virtual directories by putting their relative paths (NOT URL-encoded) in a file:

```
azcopy rm "https://[account].blob.core.windows.net/[container]/[path/to/parent/dir]" --recursive=true --list-of-files=/usr/bar/list.txt
```

Remove a single file from a Blob Storage account that has a hierarchical namespace (include/exclude not

supported):

```
azcopy rm "https://[account].dfs.core.windows.net/[container]/[path/to/file]?[SAS]"
```

Remove a single directory from a Blob Storage account that has a hierarchical namespace (include/exclude not supported):

```
azcopy rm "https://[account].dfs.core.windows.net/[container]/[path/to/directory]?[SAS]"
```

## Options

**--delete-snapshots** (string) By default, the delete operation fails if a blob has snapshots. Specify 'include' to remove the root blob and all its snapshots; alternatively specify 'only' to remove only the snapshots but keep the root blob.

**--dry-run** Prints the path files that would be removed by the command. This flag doesn't trigger the removal of the files.

**--exclude-path** (string) Exclude these paths when removing. This option doesn't support wildcard characters (\*). Checks relative path prefix. For example: myFolder;myFolder/subDirName/file.pdf

**--exclude-pattern** (string) Exclude files where the name matches the pattern list. For example:  
.jpg;.pdf;exactName

**--force-if-read-only** When deleting an Azure Files file or folder, force the deletion to work even if the existing object has its read-only attribute set

**--from-to** (string) Optionally specifies the source destination combination. For Example: BlobTrash, FileTrash, BlobFSTrash

**-h** , **--help** help for remove

**--include-path** (string) Include only these paths when removing. This option doesn't support wildcard characters (\*). Checks relative path prefix. For example: myFolder;myFolder/subDirName/file.pdf

**--include-pattern** (string) Include only files where the name matches the pattern list. For example:  
.jpg;.pdf;exactName

**--list-of-files** (string) Defines the location of a file which contains the list of files and directories to be deleted. The relative paths should be delimited by line breaks, and the paths should NOT be URL-encoded.

**--list-of-versions** (string) Specifies a file where each version ID is listed on a separate line. Ensure that the source must point to a single blob and all the version IDs specified in the file using this flag must belong to the source blob only. Specified version IDs of the given blob will get deleted from Azure Storage.

**--log-level** (string) Define the log verbosity for the log file. Available levels include: INFO(all requests/responses), WARNING(slow responses), ERROR(only failed requests), and NONE(no output logs). (default 'INFO') (default "INFO")

**--permanent-delete** (string) This is a preview feature that PERMANENTLY deletes soft-deleted snapshots/versions. Possible values include 'snapshots', 'versions', 'snapshotsandversions', 'none'. (default "none")

**--recursive** Look into subdirectories recursively when syncing between directories.

## Options inherited from parent commands

**--cap-mbps float** Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it's omitted, the throughput isn't capped.

`--output-type` (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

`--trusted-microsoft-suffixes` (string) Specifies additional domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;.core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)

# azcopy sync

8/22/2022 • 6 minutes to read • [Edit Online](#)

Replicates the source location to the destination location. This article provides a detailed reference for the azcopy sync command. To learn more about synchronizing blobs between source and destination locations, see [Synchronize with Azure Blob storage by using AzCopy v10](#). For Azure Files, see [Synchronize files](#).

## Synopsis

The last modified times are used for comparison. The file is skipped if the last modified time in the destination is more recent. The supported pairs are:

- Local <-> Azure Blob / Azure File (either SAS or OAuth authentication can be used)
- Azure Blob <-> Azure Blob (Source must include a SAS or is publicly accessible; either SAS or OAuth authentication can be used for destination)
- Azure File <-> Azure File (Source must include a SAS or is publicly accessible; SAS authentication should be used for destination)
- Azure Blob <-> Azure File

The sync command differs from the copy command in several ways:

1. By default, the recursive flag is true and sync copies all subdirectories. Sync only copies the top-level files inside a directory if the recursive flag is false.
2. When syncing between virtual directories, add a trailing slash to the path (refer to examples) if there's a blob with the same name as one of the virtual directories.
3. If the 'delete-destination' flag is set to true or prompt, then sync will delete files and blobs at the destination that aren't present at the source.

Advanced:

Note that if you don't specify a file extension, AzCopy automatically detects the content type of the files when uploading from the local disk, based on the file extension or content.

The built-in lookup table is small but on Unix it's augmented by the local system's mime.types file(s) if available under one or more of these names:

- /etc/mime.types
- /etc/apache2/mime.types
- /etc/apache/mime.types

On Windows, MIME types are extracted from the registry.

Also note that sync works off of the last modified times exclusively. So in the case of Azure File <-> Azure File, the header field Last-Modified is used instead of x-ms-file-change-time, which means that metadata changes at the source can also trigger a full copy.

```
azcopy sync [flags]
```

## Examples

Sync a single file:

```
azcopy sync "/path/to/file.txt" "https://[account].blob.core.windows.net/[container]/[path/to/blob]"
```

Same as above, but also compute an MD5 hash of the file content, and then save that MD5 hash as the blob's Content-MD5 property.

```
azcopy sync "/path/to/file.txt" "https://[account].blob.core.windows.net/[container]/[path/to/blob]" --put-md5
```

Sync an entire directory including its subdirectories (note that recursive is by default on):

```
azcopy sync "/path/to/dir" "https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]" or
azcopy sync "/path/to/dir" "https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]" --put-md5
```

Sync only the files inside of a directory but not subdirectories or the files inside of subdirectories:

```
azcopy sync "/path/to/dir" "https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]" --recursive=false
```

Sync a subset of files in a directory (For example: only jpg and pdf files, or if the file name is "exactName"):

```
azcopy sync "/path/to/dir" "https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]" --include-pattern="*.jpg;*.pdf;exactName"
```

Sync an entire directory but exclude certain files from the scope (For example: every file that starts with foo or ends with bar):

```
azcopy sync "/path/to/dir" "https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]" --exclude-pattern="foo*;*bar"
```

Sync a single blob:

```
azcopy sync "https://[account].blob.core.windows.net/[container]/[path/to/blob]?[SAS]"
"https://[account].blob.core.windows.net/[container]/[path/to/blob]"
```

Sync a virtual directory:

```
azcopy sync "https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]?[SAS]"
"https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]" --recursive=true
```

Sync a virtual directory that has the same name as a blob (add a trailing slash to the path in order to disambiguate):

```
azcopy sync "https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]/?[SAS]"
"https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]/" --recursive=true
```

Sync an Azure File directory (same syntax as Blob):

```
azcopy sync "https://[account].file.core.windows.net/[share]/[path/to/dir]?[SAS]"
"https://[account].file.core.windows.net/[share]/[path/to/dir]" --recursive=true
```

Note: if include and exclude flags are used together, only files matching the include patterns are used, but those matching the exclude patterns are ignored.

## Options

`--block-size-mb` (float) Use this block size (specified in MiB) when uploading to Azure Storage or downloading from Azure Storage. Default is automatically calculated based on file size. Decimal fractions are allowed (For example: 0.25).

`--check-md5` (string) Specifies how strictly MD5 hashes should be validated when downloading. This option is only available when downloading. Available values include: NoCheck, LogOnly, FailIfDifferent, FailIfDifferentOrMissing. (default 'FailIfDifferent'). (default "FailIfDifferent")

`--cpk-by-name` (string) Client provided key by name let clients that make requests against Azure Blob storage an

option to provide an encryption key on a per-request basis. Provided key name will be fetched from Azure Key Vault and will be used to encrypt the data

**--cpk-by-value** Client provided key by name let clients that make requests against Azure Blob storage an option to provide an encryption key on a per-request basis. Provided key and its hash will be fetched from environment variables

**--delete-destination** (string) Defines whether to delete extra files from the destination that aren't present at the source. Could be set to true, false, or prompt. If set to prompt, the user will be asked a question before scheduling files and blobs for deletion. (default 'false'). (default "false")

**--dry-run** Prints the path of files that would be copied or removed by the sync command. This flag doesn't copy or remove the actual files.

**--exclude-attributes** (string) (Windows only) Exclude files whose attributes match the attribute list. For example: A;S;R

**--exclude-path** (string) Exclude these paths when comparing the source against the destination. This option doesn't support wildcard characters (\*). Checks relative path prefix(For example: myFolder;myFolder/subDirName/file.pdf).

**--exclude-pattern** (string) Exclude files where the name matches the pattern list. For example: jpg;pdf;exactName

**--exclude-regex** (string) Exclude the relative path of the files that match with the regular expressions. Separate regular expressions with ':'.

**--from-to** (string) Optionally specifies the source destination combination. For Example: LocalBlob, BlobLocal, LocalFile, FileLocal, BlobFile, FileBlob, etc.

**-h , --help** help for sync

**--include-attributes** (string) (Windows only) Include only files whose attributes match the attribute list. For example: A;S;R

**--include-pattern** (string) Include only files where the name matches the pattern list. For example: jpg;pdf;exactName

**--include-regex** (string) Include the relative path of the files that match with the regular expressions. Separate regular expressions with ':'.

**--log-level** (string) Define the log verbosity for the log file, available levels: INFO(all requests and responses), WARNING(slow responses), ERROR(only failed requests), and NONE(no output logs). (default INFO). (default "INFO")

**--mirror-mode** Disable last-modified-time based comparison and overwrites the conflicting files and blobs at the destination if this flag is set to true. Default is false

**--preserve-permissions** False by default. Preserves ACLs between aware resources (Windows and Azure Files, or ADLS Gen 2 to ADLS Gen 2). For Hierarchical Namespace accounts, you'll need a container SAS or OAuth token with Modify Ownership and Modify Permissions permissions. For downloads, you'll also need the

**--backup** flag to restore permissions where the new Owner won't be the user running AzCopy. This flag applies to both files and folders, unless a file-only filter is specified (for example, include-pattern).

**--preserve-smb-info** For SMB-aware locations, flag will be set to true by default. Preserves SMB property info (last write time, creation time, attribute bits) between SMB-aware resources (Azure Files). This flag applies to both files and folders, unless a file-only filter is specified (for example, include-pattern). The info transferred for folders is the same as that for files, except for Last Write Time that isn't preserved for folders. (default true)

--put-md5 Create an MD5 hash of each file, and save the hash as the Content-MD5 property of the destination blob or file. (By default the hash is NOT created.) Only available when uploading.

--recursive True by default, look into subdirectories recursively when syncing between directories. (default true). (default true)

--s2s-preserve-access-tier Preserve access tier during service to service copy. Refer to [Azure Blob storage: hot, cool, and archive access tiers](#) to ensure destination storage account supports setting access tier. In the cases that setting access tier isn't supported, please use s2sPreserveAccessTier=false to bypass copying access tier. (default true). (default true)

--s2s-preserve-blob-tags Preserve index tags during service to service sync from one blob storage to another

## Options inherited from parent commands

--cap-mbps (float) Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it's omitted, the throughput isn't capped.

--output-type (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

--trusted-microsoft-suffixes (string) Specifies other domain suffixes where Azure Active Directory login tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;.core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)

# azcopy set-properties (preview)

8/22/2022 • 3 minutes to read • [Edit Online](#)

Given a location, change all the valid system properties of that storage (blob or file).

## Synopsis

```
azcopy set-properties [resourceURL] [flags]
```

Sets properties of Blob and File storage. The properties currently supported by this command are:

- Blobs -> Tier, Metadata, Tags
- Data Lake Storage Gen2 -> Tier, Metadata, Tags
- Files -> Metadata

### NOTE

Data Lake Storage Gen2 endpoints will be replaced by Blob Storage endpoints.

Refer to the examples for more information.

## Related conceptual articles

- [Get started with AzCopy](#)
- [Replace blob properties and metadata by using AzCopy v10 \(preview\)](#)

## Examples

Change tier of blob to hot:

```
azcopy set-properties "https://[account].blob.core.windows.net/[container]/[path/to/blob]" --block-blob-tier=hot
```

Change tier of blob from archive to cool with rehydrate priority set to high:

```
azcopy set-properties "https://[account].blob.core.windows.net/[container]/[path/to/blob]" --block-blob-tier=cool --rehydrate-priority=high
```

Change tier of all files in a directory to archive:

```
azcopy set-properties "https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]" --block-blob-tier=archive --recursive=true
```

Change metadata of blob to {key = "abc", val = "def"} and {key = "ghi", val = "jkl"}:

```
azcopy set-properties "https://[account].blob.core.windows.net/[container]/[path/to/blob]" --metadata=abc=def;ghi=jkl
```

Change metadata of all files in a directory to {key = "abc", val = "def"} and {key = "ghi", val = "jkl"}:

```
azcopy set-properties "https://[account].blob.core.windows.net/[container]/[path/to/virtual/dir]" --metadata=abc=def;ghi=jkl --recursive=true
```

Clear all existing metadata of blob:

```
azcopy set-properties "https://[account].blob.core.windows.net/[container]/[path/to/blob]" --metadata=clear
```

Change blob-tags of blob to {key = "abc", val = "def"} and {key = "ghi", val = "jkl"}:

```
azcopy set-properties "https://[account].blob.core.windows.net/[container]/[path/to/blob]" --blob-tags=abc=def&ghi=jkl
```

While setting tags on the blobs, there are other permissions('t' for tags) with SAS. Without those tags, the service will return an authorization error.

Clear all existing blob-tags of blob:

```
azcopy set-properties "https://[account].blob.core.windows.net/[container]/[path/to/blob]" --blob-tags=clear
```

While setting tags on the blobs, there are other permissions('t' for tags) with SAS. Without those tags, the service will return an authorization error.

## Options

- blob-tags string Set tags on blobs to categorize data in your storage account (separated by '&')
- block-blob-tier string Changes the access tier of the blobs to the given tier (default "None")
- dry-run Prints the file paths that would be affected by this command. This flag doesn't affect the actual files.
- exclude-path string Exclude these paths when removing. This option doesn't support wildcard characters (\*). Checks relative path prefix. For example: myFolder;myFolder/subDirName/file.pdf
- exclude-pattern string Exclude files where the name matches the pattern list. For example:  
.jpg;.pdf;exactName
- from-to string Optionally specifies the source destination combination. Valid values: BlobNone, FileNone, BlobFSNone
- h , --help help for set-properties
- include-path string Include only these paths when setting property. This option doesn't support wildcard characters (\*). Checks relative path prefix. For example: myFolder;myFolder/subDirName/file.pdf
- include-pattern string Include only files where the name matches the pattern list. For example:  
.jpg;.pdf;exactName
- list-of-files string Defines the location of the text file that has the list of files to be copied.
- metadata string Set the given location with these key-value pairs (separated by ';') as metadata.
- page-blob-tier string Upload page blob to Azure Storage using this blob tier. (default 'None'). (default "None")
- recursive Look into subdirectories recursively when uploading from local file system.
- rehydrate-priority string Optional flag that sets rehydrate priority for rehydration. Valid values: Standard, High. Default- standard (default "Standard")

## Options inherited from parent commands

- cap-mbps float Caps the transfer rate, in megabits per second. Moment-by-moment throughput might vary slightly from the cap. If this option is set to zero, or it's omitted, the throughput isn't capped.
- log-level (string) Define the log verbosity for the log file, available levels: INFO(all requests/responses), WARNING(slow responses), ERROR(only failed requests), and NONE(no output logs). (default 'INFO'). (default

"INFO")

--output-type (string) Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

--output-level (string) Define the output verbosity. Available levels: essential, quiet. (default "default")

--trusted-microsoft-suffixes (string) Specifies other domain suffixes where Azure Active Directory log in tokens may be sent. The default is '.core.windows.net;core.chinacloudapi.cn;core.cloudapi.de;core.usgovcloudapi.net;\*.storage.azure.net'. Any listed here are added to the default. For security, you should only put Microsoft Azure domains here. Separate multiple entries with semi-colons.

## See also

- [azcopy](#)

# Azure Policy built-in definitions for Azure Storage

8/22/2022 • 12 minutes to read • [Edit Online](#)

This page is an index of [Azure Policy](#) built-in policy definitions for Azure Storage. For additional Azure Policy built-ins for other services, see [Azure Policy built-in definitions](#).

The name of each built-in policy definition links to the policy definition in the Azure portal. Use the link in the **Version** column to view the source on the [Azure Policy GitHub repo](#).

## Microsoft.Storage

NAME (AZURE PORTAL)	DESCRIPTION	EFFECT(S)	VERSION (GITHUB)
[Preview]: [Preview]: Configure backup for blobs on storage accounts with a given tag to an existing backup vault in the same region	Enforce backup for blobs on all storage accounts that contain a given tag to a central backup vault. Doing this can help you manage backup of blobs contained across multiple storage accounts at scale. For more details, refer to <a href="https://aka.ms/AB-BlobBackupAzPolicies">https://aka.ms/AB-BlobBackupAzPolicies</a>	DeployIfNotExists, AuditIfNotExists, Disabled	<a href="#">2.0.0-preview</a>
[Preview]: [Preview]: Configure blob backup for all storage accounts that do not contain a given tag to a backup vault in the same region	Enforce backup for blobs on all storage accounts that do not contain a given tag to a central backup vault. Doing this can help you manage backup of blobs contained across multiple storage accounts at scale. For more details, refer to <a href="https://aka.ms/AB-BlobBackupAzPolicies">https://aka.ms/AB-BlobBackupAzPolicies</a>	DeployIfNotExists, AuditIfNotExists, Disabled	<a href="#">2.0.0-preview</a>
[Preview]: [Preview]: Storage account public access should be disallowed	Anonymous public read access to containers and blobs in Azure Storage is a convenient way to share data but might present security risks. To prevent data breaches caused by undesired anonymous access, Microsoft recommends preventing public access to a storage account unless your scenario requires it.	audit, Audit, deny, Deny, disabled, Disabled	<a href="#">3.1.0-preview</a>

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Azure File Sync should use private link</a>	Creating a private endpoint for the indicated Storage Sync Service resource allows you to address your Storage Sync Service resource from within the private IP address space of your organization's network, rather than through the internet-accessible public endpoint. Creating a private endpoint by itself does not disable the public endpoint.	AuditIfNotExists, Disabled	1.0.0
<a href="#">Configure Azure File Sync with private endpoints</a>	A private endpoint is deployed for the indicated Storage Sync Service resource. This enables you to address your Storage Sync Service resource from within the private IP address space of your organization's network, rather than through the internet-accessible public endpoint. The existence of one or more private endpoints by themselves does not disable the public endpoint.	DeployIfNotExists, Disabled	1.0.0
<a href="#">Configure diagnostic settings for storage accounts to Log Analytics workspace</a>	Deploys the diagnostic settings for storage accounts to stream resource logs to a Log Analytics workspace when any storage account which is missing this diagnostic settings is created or updated.	DeployIfNotExists, Disabled	1.3.0
<a href="#">Configure Storage account to use a private link connection</a>	Private endpoints connect your virtual network to Azure services without a public IP address at the source or destination. By mapping private endpoints to your storage account, you can reduce data leakage risks. Learn more about private links at - <a href="https://aka.ms/azureprivatelinkoverview">https://aka.ms/azureprivatelinkoverview</a>	DeployIfNotExists, Disabled	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Configure storage accounts to disable public network access</a>	<p>To improve the security of Storage Accounts, ensure that they aren't exposed to the public internet and can only be accessed from a private endpoint. Disable the public network access property as described in <a href="https://aka.ms/storageaccountspublicnetworkaccess">https://aka.ms/storageaccountspublicnetworkaccess</a>.</p> <p>This option disables access from any public address space outside the Azure IP range, and denies all logins that match IP or virtual network-based firewall rules. This reduces data leakage risks.</p>	Modify, Disabled	1.0.1
<a href="#">Deploy Advanced Threat Protection on storage accounts</a>	This policy enables Advanced Threat Protection on storage accounts.	DeployIfNotExists, Disabled	1.0.0
<a href="#">Geo-redundant storage should be enabled for Storage Accounts</a>	Use geo-redundancy to create highly available applications	Audit, Disabled	1.0.0
<a href="#">HPC Cache accounts should use customer-managed key for encryption</a>	Manage encryption at rest of Azure HPC Cache with customer-managed keys. By default, customer data is encrypted with service-managed keys, but customer-managed keys are commonly required to meet regulatory compliance standards. Customer-managed keys enable the data to be encrypted with an Azure Key Vault key created and owned by you. You have full control and responsibility for the key lifecycle, including rotation and management.	Audit, Disabled, Deny	2.0.0
<a href="#">Modify - Configure Azure File Sync to disable public network access</a>	The Azure File Sync's internet-accessible public endpoint are disabled by your organizational policy. You may still access the Storage Sync Service via its private endpoint(s).	Modify, Disabled	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Public network access should be disabled for Azure File Sync</a>	<p>Disabling the public endpoint allows you to restrict access to your Storage Sync Service resource to requests destined to approved private endpoints on your organization's network.</p> <p>There is nothing inherently insecure about allowing requests to the public endpoint, however, you may wish to disable it to meet regulatory, legal, or organizational policy requirements. You can disable the public endpoint for a Storage Sync Service by setting the incomingTrafficPolicy of the resource to AllowVirtualNetworksOnly.</p>	Audit, Deny, Disabled	1.0.0
<a href="#">Queue Storage should use customer-managed key for encryption</a>	<p>Secure your queue storage with greater flexibility using customer-managed keys. When you specify a customer-managed key, that key is used to protect and control access to the key that encrypts your data. Using customer-managed keys provides additional capabilities to control rotation of the key encryption key or cryptographically erase data.</p>	Audit, Deny, Disabled	1.0.0
<a href="#">Secure transfer to storage accounts should be enabled</a>	<p>Audit requirement of Secure transfer in your storage account. Secure transfer is an option that forces your storage account to accept requests only from secure connections (HTTPS). Use of HTTPS ensures authentication between the server and the service and protects data in transit from network layer attacks such as man-in-the-middle, eavesdropping, and session-hijacking</p>	Audit, Deny, Disabled	2.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Storage account containing the container with activity logs must be encrypted with BYOK</a>	This policy audits if the Storage account containing the container with activity logs is encrypted with BYOK. The policy works only if the storage account lies on the same subscription as activity logs by design. More information on Azure Storage encryption at rest can be found here <a href="https://aka.ms/azurestorage-byok">https://aka.ms/azurestorage-byok</a> .	AuditIfNotExists, Disabled	1.0.0
<a href="#">Storage account encryption scopes should use customer-managed keys to encrypt data at rest</a>	Use customer-managed keys to manage the encryption at rest of your storage account encryption scopes. Customer-managed keys enable the data to be encrypted with an Azure key-vault key created and owned by you. You have full control and responsibility for the key lifecycle, including rotation and management. Learn more about storage account encryption scopes at <a href="https://aka.ms/encryption-scopes-overview">https://aka.ms/encryption-scopes-overview</a> .	Audit, Deny, Disabled	1.0.0
<a href="#">Storage account encryption scopes should use double encryption for data at rest</a>	Enable infrastructure encryption for encryption at rest of your storage account encryption scopes for added security. Infrastructure encryption ensures that your data is encrypted twice.	Audit, Deny, Disabled	1.0.0
<a href="#">Storage account keys should not be expired</a>	Ensure the user storage account keys are not expired when key expiration policy is set, for improving security of account keys by taking action when the keys are expired.	Audit, Deny, Disabled	3.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Storage accounts should allow access from trusted Microsoft services</a>	Some Microsoft services that interact with storage accounts operate from networks that can't be granted access through network rules. To help this type of service work as intended, allow the set of trusted Microsoft services to bypass the network rules. These services will then use strong authentication to access the storage account.	Audit, Deny, Disabled	1.0.0
<a href="#">Storage accounts should be limited by allowed SKUs</a>	Restrict the set of storage account SKUs that your organization can deploy.	Audit, Deny, Disabled	1.1.0
<a href="#">Storage accounts should be migrated to new Azure Resource Manager resources</a>	Use new Azure Resource Manager for your storage accounts to provide security enhancements such as: stronger access control (RBAC), better auditing, Azure Resource Manager based deployment and governance, access to managed identities, access to key vault for secrets, Azure AD-based authentication and support for tags and resource groups for easier security management	Audit, Deny, Disabled	1.0.0
<a href="#">Storage accounts should disable public network access</a>	To improve the security of Storage Accounts, ensure that they aren't exposed to the public internet and can only be accessed from a private endpoint. Disable the public network access property as described in <a href="https://aka.ms/storageaccountpublicnetworkaccess">https://aka.ms/storageaccountpublicnetworkaccess</a> . This option disables access from any public address space outside the Azure IP range, and denies all logins that match IP or virtual network-based firewall rules. This reduces data leakage risks.	Audit, Deny, Disabled	1.0.1

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Storage accounts should have infrastructure encryption</a>	Enable infrastructure encryption for higher level of assurance that the data is secure. When infrastructure encryption is enabled, data in a storage account is encrypted twice.	Audit, Deny, Disabled	1.0.0
<a href="#">Storage accounts should have shared access signature (SAS) policies configured</a>	Ensure storage accounts have shared access signature (SAS) expiration policy enabled. Users use a SAS to delegate access to resources in Azure Storage account. And SAS expiration policy recommend upper expiration limit when a user creates a SAS token.	Audit, Deny, Disabled	1.0.0
<a href="#">Storage accounts should have the specified minimum TLS version</a>	Configure a minimum TLS version for secure communication between the client application and the storage account. To minimize security risk, the recommended minimum TLS version is the latest released version, which is currently TLS 1.2.	Audit, Deny, Disabled	1.0.0
<a href="#">Storage accounts should prevent cross tenant object replication</a>	Audit restriction of object replication for your storage account. By default, users can configure object replication with a source storage account in one Azure AD tenant and a destination account in a different tenant. It is a security concern because customer's data can be replicated to a storage account that is owned by the customer. By setting allowCrossTenantReplication to false, objects replication can be configured only if both source and destination accounts are in the same Azure AD tenant.	Audit, Deny, Disabled	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Storage accounts should prevent shared key access</a>	Audit requirement of Azure Active Directory (Azure AD) to authorize requests for your storage account. By default, requests can be authorized with either Azure Active Directory credentials, or by using the account access key for Shared Key authorization. Of these two types of authorization, Azure AD provides superior security and ease of use over Shared Key, and is recommended by Microsoft.	Audit, Deny, Disabled	1.0.0
<a href="#">Storage accounts should restrict network access</a>	Network access to storage accounts should be restricted. Configure network rules so only applications from allowed networks can access the storage account. To allow connections from specific internet or on-premises clients, access can be granted to traffic from specific Azure virtual networks or to public internet IP address ranges	Audit, Deny, Disabled	1.1.1
<a href="#">Storage accounts should restrict network access using virtual network rules</a>	Protect your storage accounts from potential threats using virtual network rules as a preferred method instead of IP-based filtering. Disabling IP-based filtering prevents public IPs from accessing your storage accounts.	Audit, Deny, Disabled	1.0.1
<a href="#">Storage Accounts should use a virtual network service endpoint</a>	This policy audits any Storage Account not configured to use a virtual network service endpoint.	Audit, Disabled	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Storage accounts should use customer-managed key for encryption</a>	<p>Secure your blob and file storage account with greater flexibility using customer-managed keys. When you specify a customer-managed key, that key is used to protect and control access to the key that encrypts your data. Using customer-managed keys provides additional capabilities to control rotation of the key encryption key or cryptographically erase data.</p>	Audit, Disabled	1.0.3
<a href="#">Storage accounts should use private link</a>	<p>Azure Private Link lets you connect your virtual network to Azure services without a public IP address at the source or destination. The Private Link platform handles the connectivity between the consumer and services over the Azure backbone network. By mapping private endpoints to your storage account, data leakage risks are reduced. Learn more about private links at - <a href="https://aka.ms/azureprivatelinkoverview">https://aka.ms/azureprivatelinkoverview</a></p>	AuditIfNotExists, Disabled	2.0.0
<a href="#">Table Storage should use customer-managed key for encryption</a>	<p>Secure your table storage with greater flexibility using customer-managed keys. When you specify a customer-managed key, that key is used to protect and control access to the key that encrypts your data. Using customer-managed keys provides additional capabilities to control rotation of the key encryption key or cryptographically erase data.</p>	Audit, Deny, Disabled	1.0.0

## Microsoft.StorageCache

NAME (AZURE PORTAL)	DESCRIPTION	EFFECT(S)	VERSION (GITHUB)
<a href="#">HPC Cache accounts should use customer-managed key for encryption</a>	Manage encryption at rest of Azure HPC Cache with customer-managed keys. By default, customer data is encrypted with service-managed keys, but customer-managed keys are commonly required to meet regulatory compliance standards. Customer-managed keys enable the data to be encrypted with an Azure Key Vault key created and owned by you. You have full control and responsibility for the key lifecycle, including rotation and management.	Audit, Disabled, Deny	<a href="#">2.0.0</a>

## Microsoft.StorageSync

NAME (AZURE PORTAL)	DESCRIPTION	EFFECT(S)	VERSION (GITHUB)
<a href="#">Azure File Sync should use private link</a>	Creating a private endpoint for the indicated Storage Sync Service resource allows you to address your Storage Sync Service resource from within the private IP address space of your organization's network, rather than through the internet-accessible public endpoint. Creating a private endpoint by itself does not disable the public endpoint.	AuditIfNotExists, Disabled	<a href="#">1.0.0</a>
<a href="#">Configure Azure File Sync with private endpoints</a>	A private endpoint is deployed for the indicated Storage Sync Service resource. This enables you to address your Storage Sync Service resource from within the private IP address space of your organization's network, rather than through the internet-accessible public endpoint. The existence of one or more private endpoints by themselves does not disable the public endpoint.	DeployIfNotExists, Disabled	<a href="#">1.0.0</a>

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Modify - Configure Azure File Sync to disable public network access</a>	The Azure File Sync's internet-accessible public endpoint are disabled by your organizational policy. You may still access the Storage Sync Service via its private endpoint(s).	Modify, Disabled	1.0.0
<a href="#">Public network access should be disabled for Azure File Sync</a>	Disabling the public endpoint allows you to restrict access to your Storage Sync Service resource to requests destined to approved private endpoints on your organization's network. There is nothing inherently insecure about allowing requests to the public endpoint, however, you may wish to disable it to meet regulatory, legal, or organizational policy requirements. You can disable the public endpoint for a Storage Sync Service by setting the incomingTrafficPolicy of the resource to AllowVirtualNetworksOnly.	Audit, Deny, Disabled	1.0.0

## Microsoft.ClassicStorage

NAME (AZURE PORTAL)	DESCRIPTION	EFFECT(S)	VERSION (GITHUB)
<a href="#">Storage accounts should be migrated to new Azure Resource Manager resources</a>	Use new Azure Resource Manager for your storage accounts to provide security enhancements such as: stronger access control (RBAC), better auditing, Azure Resource Manager based deployment and governance, access to managed identities, access to key vault for secrets, Azure AD-based authentication and support for tags and resource groups for easier security management	Audit, Deny, Disabled	1.0.0

## Next steps

- See the built-ins on the [Azure Policy GitHub repo](#).
- Review the [Azure Policy definition structure](#).

- Review [Understanding policy effects](#).

# Known issues with Azure Data Lake Storage Gen2

8/22/2022 • 4 minutes to read • [Edit Online](#)

This article describes limitations and known issues for accounts that have the hierarchical namespace feature enabled.

## NOTE

Some of the features described in this article might not be supported in accounts that have Network File System (NFS) 3.0 support enabled. To view a table that shows the impact of feature support when various capabilities are enabled, see [Blob Storage feature support in Azure Storage accounts](#).

## Supported Blob storage features

An increasing number of Blob storage features now work with accounts that have a hierarchical namespace. For a complete list, see [Blob Storage features available in Azure Data Lake Storage Gen2](#).

## Supported Azure service integrations

Azure Data Lake Storage Gen2 supports several Azure services that you can use to ingest data, perform analytics, and create visual representations. For a list of supported Azure services, see [Azure services that support Azure Data Lake Storage Gen2](#).

For more information, see [Azure services that support Azure Data Lake Storage Gen2](#).

## Supported open source platforms

Several open source platforms support Data Lake Storage Gen2. For a complete list, see [Open source platforms that support Azure Data Lake Storage Gen2](#).

For more information, see [Open source platforms that support Azure Data Lake Storage Gen2](#).

## Blob storage APIs

Data Lake Storage Gen2 APIs, NFS 3.0, and Blob APIs can operate on the same data.

This section describes issues and limitations with using blob APIs, NFS 3.0, and Data Lake Storage Gen2 APIs to operate on the same data.

- You cannot use blob APIs, NFS 3.0, and Data Lake Storage APIs to write to the same instance of a file. If you write to a file by using Data Lake Storage Gen2 or APIs or NFS 3.0, then that file's blocks won't be visible to calls to the [Get Block List](#) blob API. The only exception is when using you are overwriting. You can overwrite a file/blob using either API or with NFS 3.0 by using the zero-truncate option.
- When you use the [List Blobs](#) operation without specifying a delimiter, the results will include both directories and blobs. If you choose to use a delimiter, use only a forward slash ( / ). This is the only supported delimiter.
- If you use the [Delete Blob](#) API to delete a directory, that directory will be deleted only if it's empty. This means that you can't use the Blob API delete directories recursively.

These Blob REST APIs aren't supported:

- [Put Blob \(Page\)](#)
- [Put Page](#)
- [Get Page Ranges](#)
- [Incremental Copy Blob](#)
- [Put Page from URL](#)

Unmanaged VM disks are not supported in accounts that have a hierarchical namespace. If you want to enable a hierarchical namespace on a storage account, place unmanaged VM disks into a storage account that doesn't have the hierarchical namespace feature enabled.

## Support for setting access control lists (ACLs) recursively

The ability to apply ACL changes recursively from parent directory to child items is generally available. In the current release of this capability, you can apply ACL changes by using Azure Storage Explorer, PowerShell, Azure CLI, and the .NET, Java, and Python SDK. Support is not yet available for the Azure portal.

## Access control lists (ACL) and anonymous read access

If [anonymous read access](#) has been granted to a container, then ACLs have no effect on that container or the files in that container. This only affects read requests. Write requests will still honor the ACLs.

## AzCopy

Use only the latest version of AzCopy ([AzCopy v10](#)). Earlier versions of AzCopy such as AzCopy v8.1, are not supported.

## Azure Storage Explorer

Use only versions [1.6.0](#) or higher.

## Storage Explorer in the Azure portal

ACLs are not yet supported.

## Third party applications

Third party applications that use REST APIs to work will continue to work if you use them with Data Lake Storage Gen2. Applications that call Blob APIs will likely work.

## Storage Analytics logs (classic)

The setting for retention days is not yet supported, but you can delete logs manually by using any supported tool such as Azure Storage Explorer, REST or an SDK.

## Windows Azure Storage Blob (WASB) driver

Currently, the WASB driver, which was designed to work with the Blob API only, encounters problems in a few common scenarios. Specifically, when it is a client to a hierarchical namespace enabled storage account. Multi-protocol access on Data Lake Storage won't mitigate these issues.

Using the WASB driver as a client to a hierarchical namespace enabled storage account is not supported.

Instead, we recommend that you use the [Azure Blob File System \(ABFS\)](#) driver in your Hadoop environment. If you are trying to migrate off of an on-premises Hadoop environment with a version earlier than Hadoop branch-3, then please open an Azure Support ticket so that we can get in touch with you on the right path forward for you and your organization.

## Soft delete for blobs capability

If parent directories for soft-deleted files or directories are renamed, the soft-deleted items may not be displayed correctly in the Azure portal. In such cases you can use [PowerShell](#) or [Azure CLI](#) to list and restore the soft-deleted items.

## Events

If your account has an event subscription, read operations on the secondary endpoint will result in an error. To resolve this issue, remove event subscriptions.

### TIP

Read access to the secondary endpoint is available only when you enable read-access geo-redundant storage (RA-GRS) or read-access geo-zone-redundant storage (RA-GZRS).

# Microsoft client tools for working with Azure Storage

8/22/2022 • 2 minutes to read • [Edit Online](#)

Microsoft provides several graphical user interface (GUI) tools for working with the data in your Azure Storage account. All of the tools outlined in the following table are free.

AZURE STORAGE CLIENT TOOL	SUPPORTED PLATFORMS	BLOCK BLOB	PAGE BLOB	APPEND BLOB	TABLES	QUEUES	FILES
<a href="#">Azure portal</a>	Web	Yes	Yes	Yes	Yes	Yes	Yes
<a href="#">Azure Storage Explorer</a>	Windows, OSX	Yes	Yes	Yes	Yes	Yes	Yes
<a href="#">Microsoft Visual Studio Cloud Explorer</a>	Windows	Yes	Yes	Yes	Yes	Yes	No

There are also a number of third-party tools available for working with Azure Storage data.

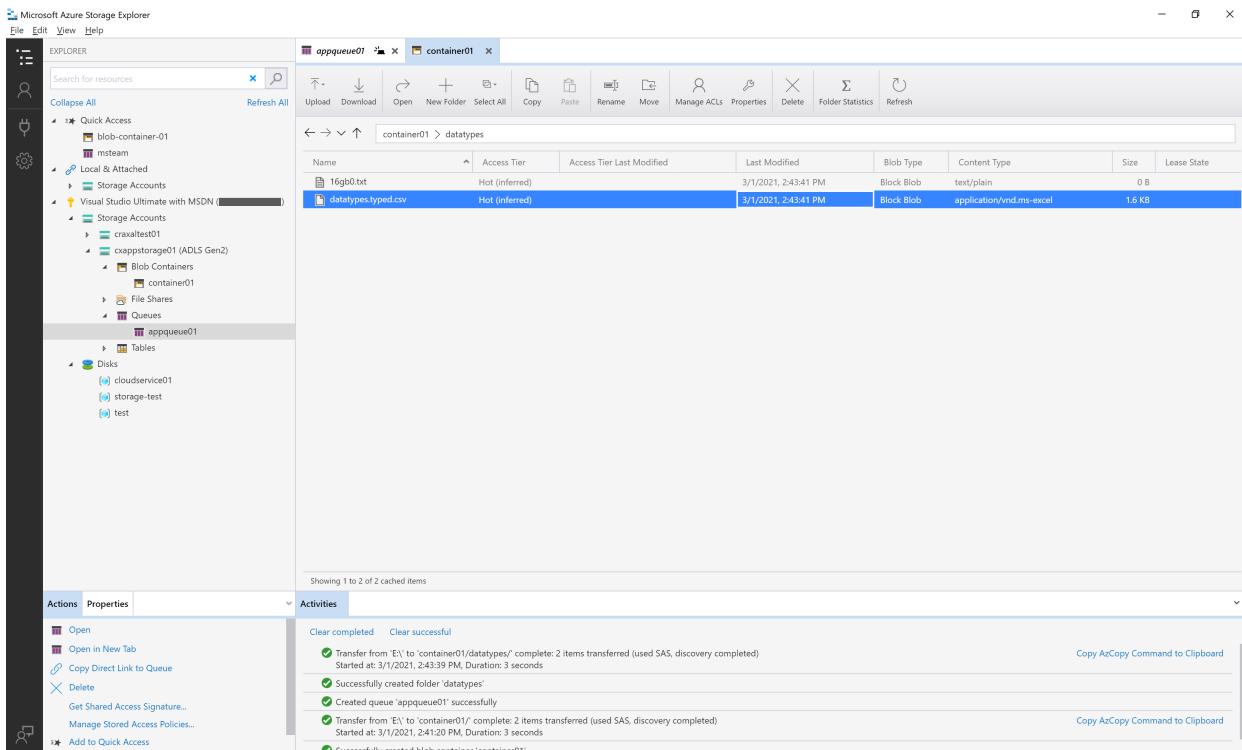
# Get started with Storage Explorer

8/22/2022 • 8 minutes to read • [Edit Online](#)

## Overview

Microsoft Azure Storage Explorer is a standalone app that makes it easy to work with Azure Storage data on Windows, macOS, and Linux.

In this article, you'll learn several ways of connecting to and managing your Azure storage accounts.



## Prerequisites

- [Windows](#)
- [macOS](#)
- [Linux](#)

The following versions of Windows support Storage Explorer:

- Windows 11
- Windows 10
- Windows 8
- Windows 7

For all versions of Windows, Storage Explorer requires .NET Framework 4.7.2 at a minimum.

## Download and install

To download and install Storage Explorer, see [Azure Storage Explorer](#).

## Connect to a storage account or service

Storage Explorer provides several ways to connect to Azure resources:

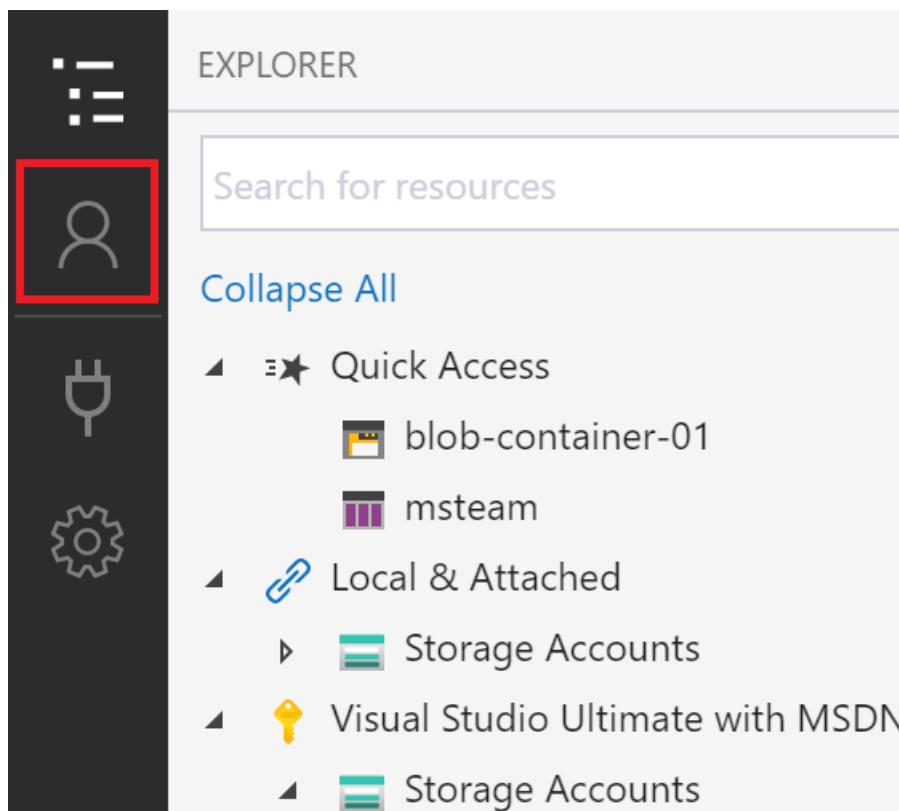
- [Sign in to Azure to access your subscriptions and their resources](#)
- [Attach to an individual Azure Storage resource](#)

## Sign in to Azure

### NOTE

To fully access resources after you sign in, Storage Explorer requires both management (Azure Resource Manager) and data layer permissions. This means that you need Azure Active Directory (Azure AD) permissions to access your storage account, the containers in the account, and the data in the containers. If you have permissions only at the data layer, consider choosing the **Sign in using Azure Active Directory (Azure AD)** option when attaching to a resource. For more information about the specific permissions Storage Explorer requires, see the [Azure Storage Explorer troubleshooting guide](#).

1. In Storage Explorer, select **View > Account Management** or select the **Manage Accounts** button.



2. **ACCOUNT MANAGEMENT** now displays all the Azure accounts you're signed in to. To connect to another account, select **Add an account....**
3. The **Connect to Azure Storage** dialog opens. In the **Select Resource** panel, select **Subscription**.

## Select Resource

Select Resource > Authenticate > Connect

What kind of Azure resource do you want to connect to?

-  **Subscription**  
Sign in to Azure to access storage resources such as blobs, files, queues, and tables under subscriptions you have access to.
-  **Storage account**  
Attach to a Storage account.
-  **Blob container**  
Attach to a Blob container.
-  **ADLS Gen2 container or directory**  
Attach to an ADLS Gen2 container or directory.
-  **File share**  
Attach to a File share.
-  **Queue**  
Attach to a queue.
-  **Table**  
Attach to a table.
-  **Local storage emulator**  
Attach to resources managed by a storage emulator running on your local machine.

[Cancel](#)

4. In the **Select Azure Environment** panel, select an Azure environment to sign in to. You can sign in to global Azure, a national cloud or an Azure Stack instance. Then select **Next**.

## Select Azure Environment

Select Resource > Select Azure Environment > Sign In

Which Azure environment will you use to sign in?

- Azure
- Azure China
- Azure Germany
- Azure US Government
- Custom Environment:

test

[Manage custom environments...](#)

[Back](#)

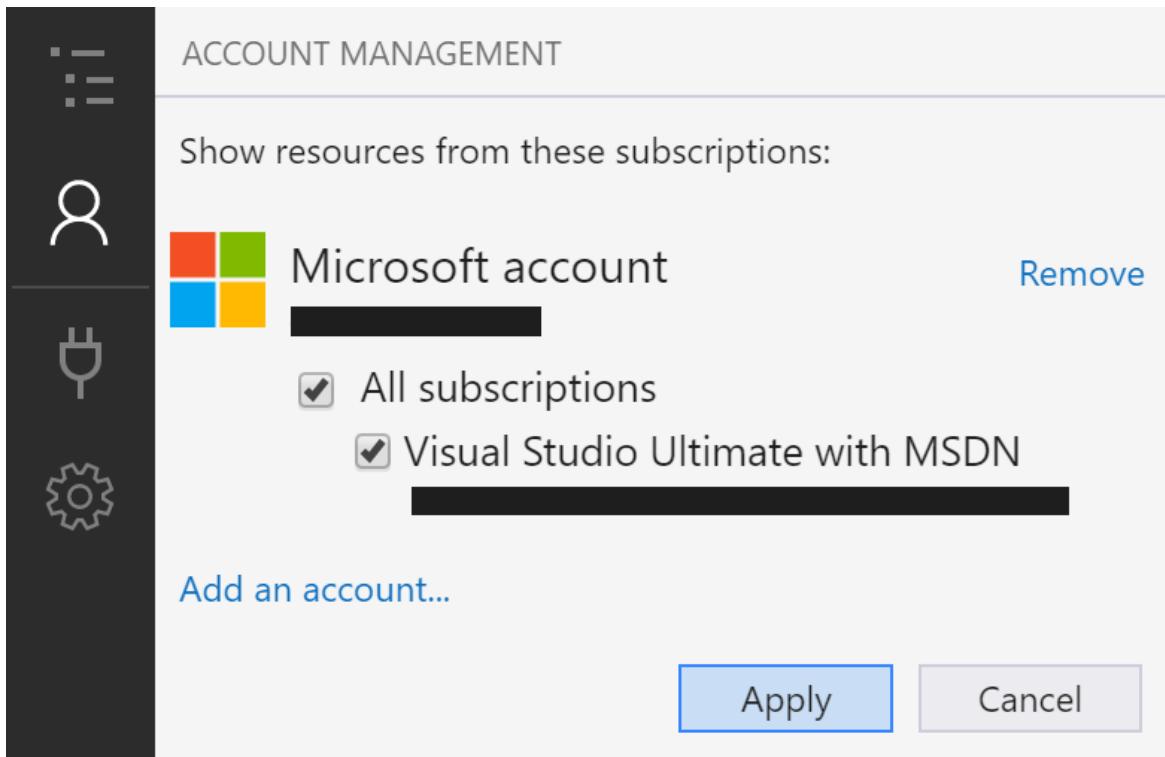
[Next](#)

[Cancel](#)

**TIP**

For more information about Azure Stack, see [Connect Storage Explorer to an Azure Stack subscription or storage account](#).

5. Storage Explorer will open a webpage for you to sign in.
6. After you successfully sign in with an Azure account, the account and the Azure subscriptions associated with that account appear under **ACCOUNT MANAGEMENT**. Select the Azure subscriptions that you want to work with, and then select **Apply**.



7. EXPLORER displays the storage accounts associated with the selected Azure subscriptions.

The screenshot shows the Azure Storage Explorer interface. On the left is a sidebar with icons for Quick Access, Local & Attached, and Storage Accounts. The main area has a search bar at the top. Below it, there are buttons for 'Collapse All' and 'Refresh All'. The main content area displays a hierarchical list of storage resources. A red box highlights the 'Visual Studio Ultimate with MSDN' account node, which is expanded to show its contents: 'Storage Accounts' and 'Disks'. Under 'Storage Accounts', there are three entries: 'craxaltest01', 'cxappstorage01 (ADLS Gen2)', and 'test'. Under 'Disks', there are three entries: 'cloudservice01', 'storage-test', and 'test'.

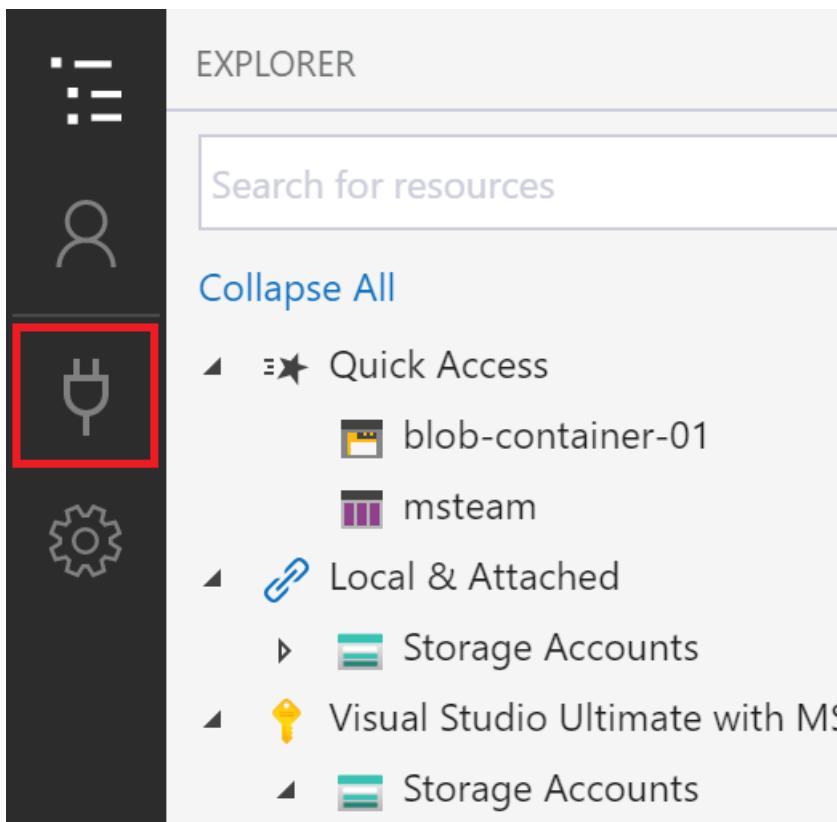
#### Attach to an individual resource

Storage Explorer lets you connect to individual resources, such as an Azure Data Lake Storage Gen2 container, using various authentication methods. Some authentication methods are only supported for certain resource types.

RESOURCE TYPE	AZURE AD	ACCOUNT NAME AND KEY	SHARED ACCESS SIGNATURE (SAS)	PUBLIC (ANONYMOUS)
Storage accounts	Yes	Yes	Yes (connection string or URL)	No
Blob containers	Yes	No	Yes (URL)	Yes
Gen2 containers	Yes	No	Yes (URL)	Yes
Gen2 directories	Yes	No	Yes (URL)	Yes
File shares	No	No	Yes (URL)	No
Queues	Yes	No	Yes (URL)	No
Tables	Yes	No	Yes (URL)	No

Storage Explorer can also connect to a [local storage emulator](#) using the emulator's configured ports.

To connect to an individual resource, select the **Connect** button in the left-hand toolbar. Then follow the instructions for the resource type you want to connect to.



When a connection to a storage account is successfully added, a new tree node will appear under **Local & Attached > Storage Accounts**.

For other resource types, a new node is added under **Local & Attached > Storage Accounts > (Attached Containers)**. The node will appear under a group node matching its type. For example, a new connection to an Azure Data Lake Storage Gen2 container will appear under **Blob Containers**.

If Storage Explorer couldn't add your connection, or if you can't access your data after successfully adding the connection, see the [Azure Storage Explorer troubleshooting guide](#).

The following sections describe the different authentication methods you can use to connect to individual resources.

#### Azure AD

Storage Explorer can use your Azure account to connect to the following resource types:

- Blob containers
- Azure Data Lake Storage Gen2 containers
- Azure Data Lake Storage Gen2 directories
- Queues

Azure AD is the preferred option if you have data layer access to your resource but no management layer access.

1. Sign in to at least one Azure account using the [steps described above](#).
2. In the **Select Resource** panel of the **Connect to Azure Storage** dialog, select **Blob container**, **ADLS Gen2 container**, or **Queue**.
3. Select **Sign in using Azure Active Directory (Azure AD)** and select **Next**.
4. Select an Azure account and tenant. The account and tenant must have access to the Storage resource you

want to attach to. Select **Next**.

5. Enter a display name for your connection and the URL of the resource. Select **Next**.
6. Review your connection information in the **Summary** panel. If the connection information is correct, select **Connect**.

#### Account name and key

Storage Explorer can connect to a storage account using the storage account's name and key.

You can find your account keys in the [Azure portal](#). Open your storage account page and select **Settings > Access keys**.

1. In the **Select Resource** panel of the **Connect to Azure Storage** dialog, select **Storage account**.
2. Select **Account name and key** and select **Next**.
3. Enter a display name for your connection, the name of the account, and one of the account keys. Select the appropriate Azure environment. Select **Next**.
4. Review your connection information in the **Summary** panel. If the connection information is correct, select **Connect**.

#### Shared access signature (SAS) connection string

Storage Explorer can connect to a storage account using a connection string with a Shared Access Signature (SAS). A SAS connection string looks like this:

```
SharedAccessSignature=sv=2020-04-08&ss=btqf&srt=sco&st=2021-03-02T00%3A22%3A19Z&se=2020-03-03T00%3A22%3A19Z&sp=r&sig=fFFpX%2F5tzqmmFfaL0wRffH1hfFFLn6zJuy1T6yhOo%2FY%3F;
BlobEndpoint=https://contoso.blob.core.windows.net/;
FileEndpoint=https://contoso.file.core.windows.net/;
QueueEndpoint=https://contoso.queue.core.windows.net/;
TableEndpoint=https://contoso.table.core.windows.net/;
```

1. In the **Select Resource** panel of the **Connect to Azure Storage** dialog, select **Storage account**.
2. Select **Shared access signature (SAS)** and select **Next**.
3. Enter a display name for your connection and the SAS connection string for the storage account. Select **Next**.
4. Review your connection information in the **Summary** panel. If the connection information is correct, select **Connect**.

#### Shared access signature (SAS) URL

Storage Explorer can connect to the following resource types using a SAS URI:

- Blob container
- Azure Data Lake Storage Gen2 container or directory
- File share
- Queue
- Table

A SAS URI looks like this:

```
https://contoso.blob.core.windows.net/container01?sv=2020-04-08&st=2021-03-02T00%3A30%3A33Z&se=2020-03-03T00%3A30%3A33Z&sr=c&sp=r&sig=z9VFDWffrV6FXU51T8b8HVfipZP0pYOFLXuQw6wfkFY%3F
```

1. In the **Select Resource** panel of the **Connect to Azure Storage** dialog, select the resource you want to connect to.
2. Select **Shared access signature (SAS)** and select **Next**.
3. Enter a display name for your connection and the SAS URI for the resource. Select **Next**.
4. Review your connection information in the **Summary** panel. If the connection information is correct, select

## Connect.

### Local storage emulator

Storage Explorer can connect to an Azure Storage emulator. Currently, there are two supported emulators:

- [Azure Storage Emulator](#) (Windows only)
- [Azurite](#) (Windows, macOS, or Linux)

If your emulator is listening on the default ports, you can use the **Local & Attached > Storage Accounts > Emulator - Default Ports** node to access your emulator.

If you want to use a different name for your connection, or if your emulator isn't running on the default ports:

1. Start your emulator.

#### IMPORTANT

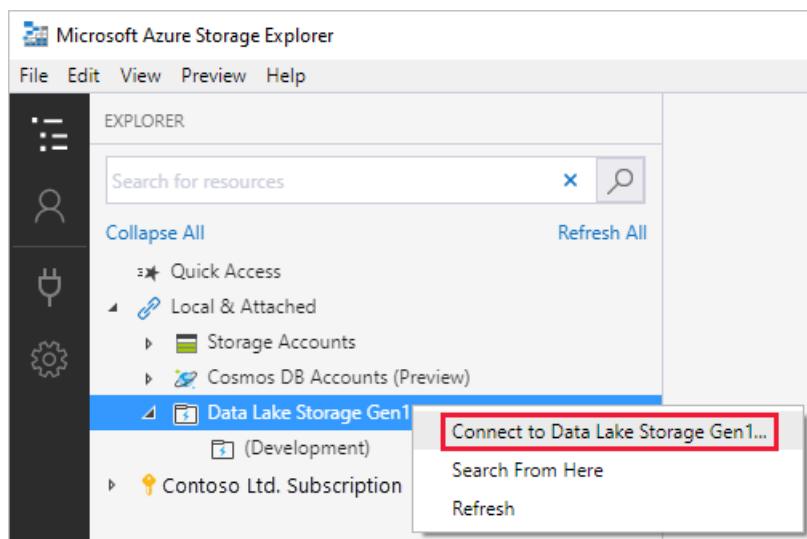
Storage Explorer doesn't automatically start your emulator. You must start it manually.

2. In the **Select Resource** panel of the **Connect to Azure Storage** dialog, select **Local storage emulator**.
3. Enter a display name for your connection and the port number for each emulated service you want to use. If you don't want to use to a service, leave the corresponding port blank. Select **Next**.
4. Review your connection information in the **Summary** panel. If the connection information is correct, select **Connect**.

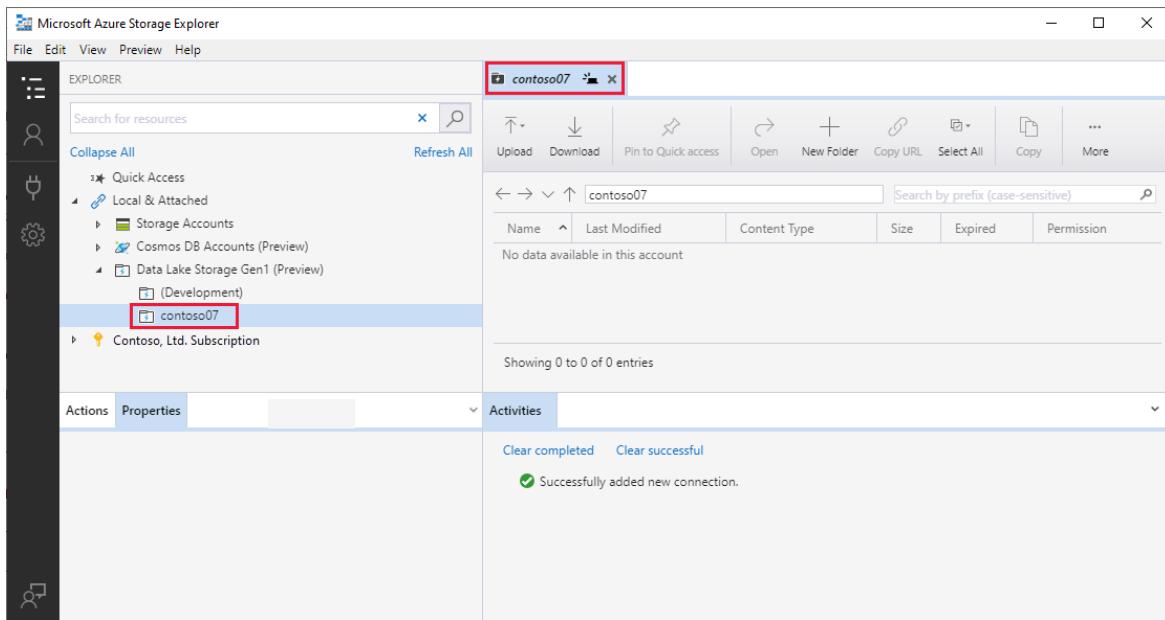
### Connect to Azure Data Lake Store by URI

You can access a resource that's not in your subscription. You need someone who has access to that resource to give you the resource URI. After you sign in, connect to Data Lake Store by using the URI. To connect, follow these steps:

1. Under **EXPLORER**, expand **Local & Attached**.
2. Right-click **Data Lake Storage Gen1**, and select **Connect to Data Lake Storage Gen1**.



3. Enter the URI, and then select **OK**. Your Data Lake Store appears under **Data Lake Storage**.

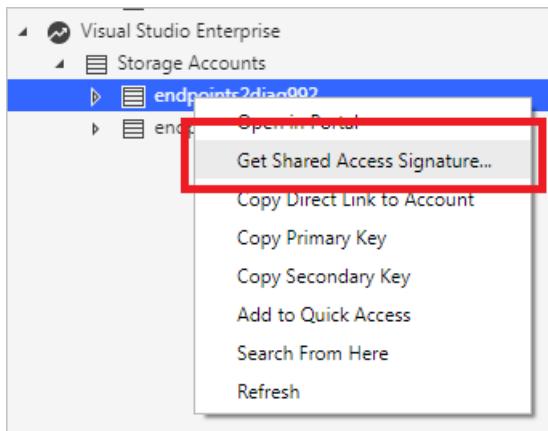


This example uses Data Lake Storage Gen1. Azure Data Lake Storage Gen2 is now available. For more information, see [What is Azure Data Lake Storage Gen1](#).

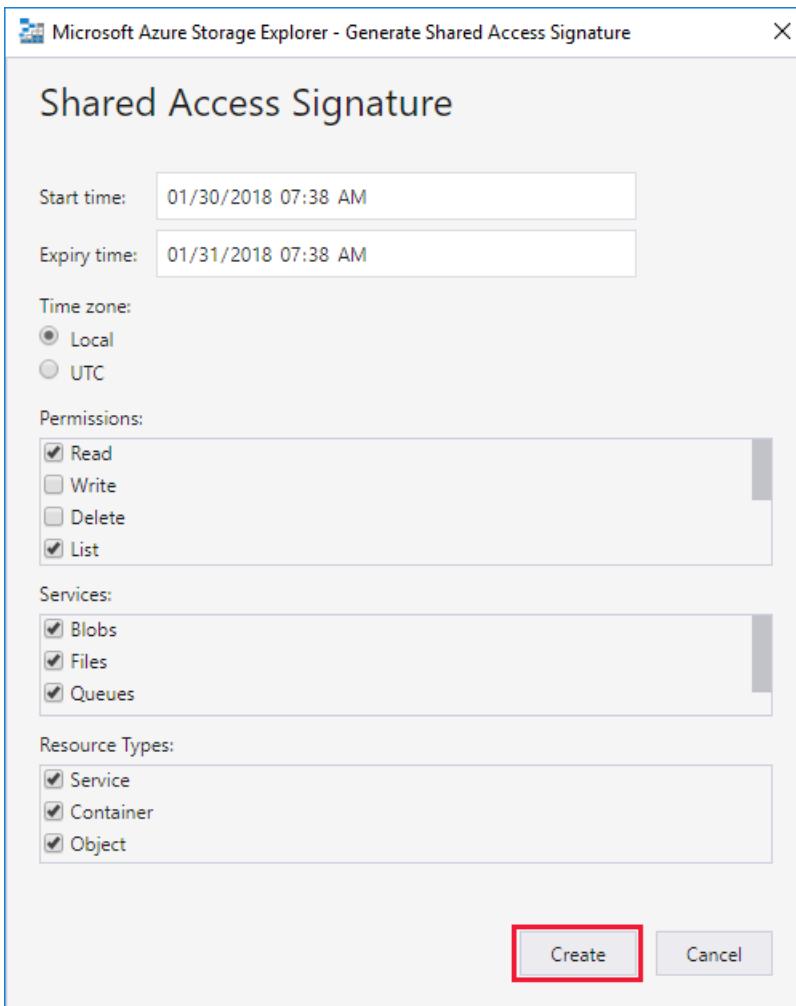
## Generate a shared access signature in Storage Explorer

### Account level shared access signature

1. Right-click the storage account you want share, and then select Get Shared Access Signature.



2. In Shared Access Signature, specify the time frame and permissions you want for the account, and then select Create.



3. Copy either the **Connection string** or the raw **Query string** to your clipboard.

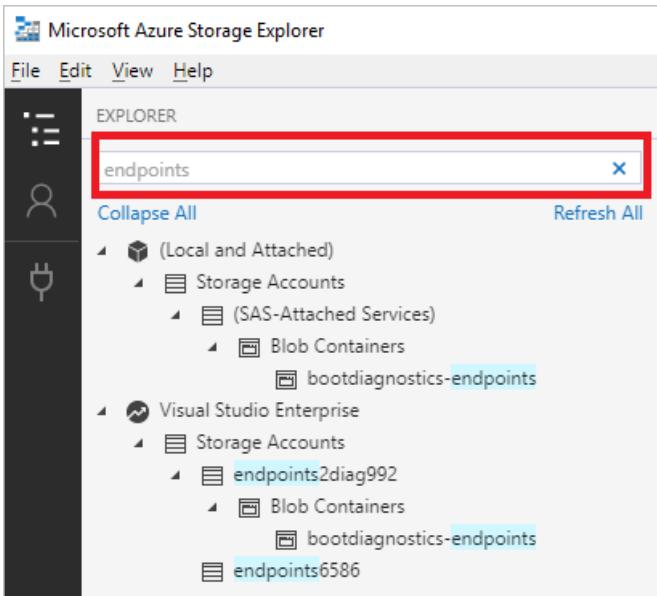
#### Service level shared access signature

You can get a shared access signature at the service level. For more information, see [Get the SAS for a blob container](#).

## Search for storage accounts

To find a storage resource, you can search in the **EXPLORER** pane.

As you enter text in the search box, Storage Explorer displays all resources that match the search value you've entered up to that point. This example shows a search for **endpoints**:



#### NOTE

To speed up your search, use **Account Management** to deselect any subscriptions that don't contain the item you're searching for. You can also right-click a node and select **Search From Here** to start searching from a specific node.

## Next steps

- [Manage Azure Blob storage resources with Storage Explorer](#)
- [Manage Azure Data Lake Store resources with Storage Explorer](#)

# Azure Storage Explorer troubleshooting guide

8/22/2022 • 24 minutes to read • [Edit Online](#)

Microsoft Azure Storage Explorer is a standalone app that makes it easy to work with Azure Storage data on Windows, macOS, and Linux. The app can connect to storage accounts hosted on Azure, national clouds, and Azure Stack.

This guide summarizes solutions for issues that are commonly seen in Storage Explorer.

## Azure RBAC permissions issues

Azure role-based access control ([Azure RBAC](#)) enables highly granular access management of Azure resources by combining sets of permissions into *roles*. Here are some strategies to get Azure RBAC working optimally in Storage Explorer.

### How do I access my resources in Storage Explorer?

If you're having problems accessing storage resources through Azure RBAC, you might not have been assigned the appropriate roles. The following sections describe the permissions Storage Explorer currently requires for access to your storage resources. Contact your Azure account admin if you're not sure you have the appropriate roles or permissions.

#### "Read: List/Get Storage Account(s)" permissions issue

You must have permission to list storage accounts. To get this permission, you must be assigned the *Reader* role.

#### List storage account keys

Storage Explorer can also use account keys to authenticate requests. You can get access to account keys through more powerful roles, such as the *Contributor* role.

#### NOTE

Access keys grant unrestricted permissions to anyone who holds them. As a result, we don't recommend that you hand out these keys to account users. If you need to revoke access keys, you can regenerate them from the [Azure portal](#).

#### Data roles

You must be assigned at least one role that grants access to read data from resources. For example, if you want to list or download blobs, you'll need at least the *Storage Blob Data Reader* role.

### Why do I need a management layer role to see my resources in Storage Explorer?

Azure Storage has two layers of access: *management* and *data*. Subscriptions and storage accounts are accessed through the management layer. Containers, blobs, and other data resources are accessed through the data layer. For example, if you want to get a list of your storage accounts from Azure, you send a request to the management endpoint. If you want a list of blob containers in an account, you send a request to the appropriate service endpoint.

Azure roles can grant you permissions for management or data layer access. The Reader role, for example, grants read-only access to management layer resources.

Strictly speaking, the Reader role provides no data layer permissions and isn't necessary for accessing the data layer.

Storage Explorer makes it easy to access your resources by gathering the necessary information to connect to your Azure resources. For example, to display your blob containers, Storage Explorer sends a "list containers"

request to the blob service endpoint. To get that endpoint, Storage Explorer searches the list of subscriptions and storage accounts you have access to. To find your subscriptions and storage accounts, Storage Explorer also needs access to the management layer.

If you don't have a role that grants any management layer permissions, Storage Explorer can't get the information it needs to connect to the data layer.

### What if I can't get the management layer permissions I need from my admin?

If you want to access blob containers, Azure Data Lake Storage Gen2 containers or directories, or queues, you can attach to those resources by using your Azure credentials.

1. Open the **Connect** dialog.
2. Select the resource type you want to connect to.
3. Select **Sign in using Azure Active Directory (Azure AD)** and select **Next**.
4. Select the user account and tenant associated with the resource you're attaching to. Select **Next**.
5. Enter the URL to the resource, and enter a unique display name for the connection. Select **Next** and then select **Connect**.

For other resource types, we don't currently have an Azure RBAC-related solution. As a workaround, you can request a shared access signature URL and then attach to your resource:

1. Open the **Connect** dialog.
2. Select the resource type you want to connect to.
3. Select **Shared access signature (SAS)** and select **Next**.
4. Enter the shared access signature URL you received and enter a unique display name for the connection. Select **Next** and then select **Connect**.

For more information on how to attach to resources, see [Attach to an individual resource](#).

### Recommended Azure built-in roles

There are several Azure built-in roles that can provide the permissions needed to use Storage Explorer. Some of those roles are:

- **Owner**: Manage everything, including access to resources.
- **Contributor**: Manage everything, excluding access to resources.
- **Reader**: Read and list resources.
- **Storage Account Contributor**: Full management of storage accounts.
- **Storage Blob Data Owner**: Full access to Azure Storage blob containers and data.
- **Storage Blob Data Contributor**: Read, write, and delete Azure Storage containers and blobs.
- **Storage Blob Data Reader**: Read and list Azure Storage containers and blobs.

#### NOTE

The Owner, Contributor, and Storage Account Contributor roles grant account key access.

## SSL certificate issues

This section discusses SSL certificate issues.

### Understand SSL certificate issues

Make sure you've read the [SSL certificates section](#) in the Storage Explorer networking documentation before you continue.

### Use system proxy

If you're only using features that support the **use system proxy** setting, try using that setting. To read more about the **system proxy** setting, see [Network connections in Storage Explorer](#).

## Import SSL certificates

If you have a copy of the self-signed certificates, you can instruct Storage Explorer to trust them:

1. Obtain a Base-64 encoded X.509 (.cer) copy of the certificate.
2. Go to **Edit > SSL Certificates > Import Certificates**. Then use the file picker to find, select, and open the .cer file.

This issue might also occur if there are multiple certificates (root and intermediate). To fix this error, all certificates must be imported.

## Find SSL certificates

If you don't have a copy of the self-signed certificates, talk to your IT admin for help.

Follow these steps to find them:

1. Install OpenSSL:
  - **Windows**: Any of the light versions should be sufficient.
  - Mac and Linux: Should be included with your operating system.
2. Run OpenSSL:
  - Windows: Open the installation directory, select `/bin/`, and then double-click `openssl.exe`.
  - Mac and Linux: Run `openssl` from a terminal.
3. Run the command `openssl s_client -showcerts -connect <hostname>:443` for any of the Microsoft or Azure host names that your storage resources are behind. For more information, see this [list of host names that are frequently accessed by Storage Explorer](#).
4. Look for self-signed certificates. If the subject `("s:")` and issuer `("i:")` are the same, the certificate is most likely self-signed.
5. When you find the self-signed certificates, for each one, copy and paste everything from, and including, `-----BEGIN CERTIFICATE-----` to `-----END CERTIFICATE-----` into a new .cer file.
6. Open Storage Explorer and go to **Edit > SSL Certificates > Import Certificates**. Then use the file picker to find, select, and open the .cer files you created.

## Disable SSL certificate validation

If you can't find any self-signed certificates by following these steps, contact us through the feedback tool. You can also open Storage Explorer from the command line with the `--ignore-certificate-errors` flag. When opened with this flag, Storage Explorer ignores certificate errors. *This flag is not recommended.*

# Sign-in issues

This section discusses sign-in issues you might encounter.

## Understand sign-in

Make sure you've read the [Sign in to Storage Explorer](#) documentation before you continue.

## Frequently having to reenter credentials

Having to reenter credentials is most likely the result of Conditional Access policies set by your Azure Active Directory (Azure AD) admin. When Storage Explorer asks you to reenter credentials from the account panel, you should see an **Error details** link. Select it to see why Storage Explorer is asking you to reenter credentials. Conditional Access policy errors that require reentering of credentials might look something like these:

- The refresh token has expired.
- You must use multifactor authentication to access.
- Your admin made a configuration change.

To reduce the frequency of having to reenter credentials because of errors like the preceding ones, you'll need to talk to your Azure AD admin.

### Conditional access policies

If you have conditional access policies that need to be satisfied for your account, make sure you're using the **Default Web Browser** value for the **Sign in with** setting. For information on that setting, see [Changing where sign-in happens](#).

### Browser complains about HTTP redirect or insecure connection during sign-in

When Storage Explorer performs sign-in in your web browser, a redirect to `localhost` is done at the end of the sign-in process. Browsers sometimes raise a warning or error that the redirect is being performed with HTTP instead of HTTPS. Some browsers might also try to force the redirect to be performed with HTTPS. If either of these issues happen, depending on your browser, you have options:

- Ignore the warning.
- Add an exception for `localhost`.
- Disable force HTTPS, either globally or just for `localhost`.

If you can't do any of those options, you can also [change where sign-in happens](#) to integrated sign-in to avoid using your browser altogether.

### Unable to acquire token, tenant is filtered out

If you see an error message that says a token can't be acquired because a tenant is filtered out, you're trying to access a resource that's in a tenant you filtered out. To unfilter the tenant, go to the **Account Panel**. Make sure the checkbox for the tenant specified in the error is selected. For more information on filtering tenants in Storage Explorer, see [Managing accounts](#).

### Authentication library failed to start properly

If on startup you see an error message that says Storage Explorer's authentication library failed to start properly, make sure your installation environment meets all [prerequisites](#). Not meeting prerequisites is the most likely cause of this error message.

If you believe that your installation environment meets all prerequisites, [open an issue on GitHub](#). When you open your issue, make sure to include:

- Your OS.
- What version of Storage Explorer you're trying to use.
- If you checked the prerequisites.
- [Authentication logs](#) from an unsuccessful launch of Storage Explorer. Verbose authentication logging is automatically enabled after this type of error occurs.

### Blank window when you use integrated sign-in

If you chose to use **Integrated Sign-in** and you're seeing a blank sign-in window, you'll likely need to switch to a different sign-in method. Blank sign-in dialog boxes most often occur when an Active Directory Federation Services server prompts Storage Explorer to perform a redirect that's unsupported by Electron.

To change to a different sign-in method, change the **Sign in with** setting under **Settings > Application > Sign-in**. For information on the different types of sign-in methods, see [Changing where sign in happens](#).

### Reauthentication loop or UPN change

If you're in a reauthentication loop or have changed the UPN of one of your accounts, try these steps:

1. Open Storage Explorer.
2. Go to **Help > Reset**.
3. Make sure at least **Authentication** is selected. Clear other items you don't want to reset.
4. Select **Reset**.
5. Restart Storage Explorer and try to sign in again.

If you continue to have issues after you do a reset, try these steps:

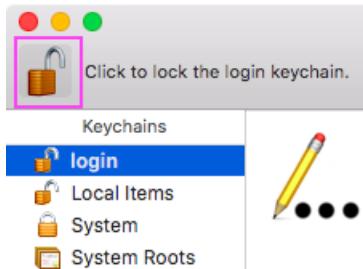
1. Open Storage Explorer.
2. Remove all accounts and then close Storage Explorer.
3. Delete the `./identityService` folder from your machine. On Windows, the folder is located at `C:\users\<username>\AppData\Local`. For Mac and Linux, you can find the folder at the root of your user directory.
4. If you're running Mac or Linux, you also need to delete the `Microsoft.Developer.IdentityService` entry from your operating system's keystore. On the Mac, the keystore is the Gnome Keychain application. In Linux, the application is typically called `Keyring`, but the name might differ depending on your distribution.
5. Restart Storage Explorer and try to sign in again.

#### **macOS: Keychain errors or no sign-in window**

macOS Keychain can sometimes enter a state that causes issues for the Storage Explorer authentication library.

To get Keychain out of this state:

1. Close Storage Explorer.
2. Open Keychain by selecting **Command+Spacebar**, enter **keychain**, and select **Enter**.
3. Select the **login** keychain.
4. Select the **padlock** to lock the keychain. After the process is finished, the **padlock** appears locked. It might take a few seconds, depending on what apps you have open.



5. Open Storage Explorer.
6. You're prompted with a message like "Service hub wants to access the Keychain." Enter your Mac admin account password and select **Always Allow**. Or select **Allow** if **Always Allow** isn't available.
7. Try to sign in.

#### **Default browser doesn't open**

If your default browser doesn't open when you try to sign in, try all of the following techniques:

- Restart Storage Explorer.
- Open your browser manually before you start to sign in.
- Try using **Integrated Sign-In**. For instructions, see [Changing where sign-in happens](#).

#### **Other sign-in issues**

If none of the preceding instructions apply to your sign-in issue or if they fail to resolve your sign-in issue, [open an issue on GitHub](#).

#### **Missing subscriptions and broken tenants**

If you can't retrieve your subscriptions after you successfully sign in, try the following troubleshooting methods:

- Verify that your account has access to the subscriptions you expect. You can verify your access by signing in to the portal for the Azure environment you're trying to use.
- Make sure you've signed in through the correct Azure environment like Azure, Azure China 21Vianet, Azure Germany, Azure US Government, or Custom Environment.
- If you're behind a proxy server, make sure you configured the Storage Explorer proxy correctly.
- Try removing and re-adding the account.
- If there's a "More information" or "Error details" link, check which error messages are being reported for the tenants that are failing. If you aren't sure how to respond to the error messages, [open an issue in GitHub](#).

## Problem interacting with your OS credential store during an AzCopy transfer

If you see this message on Windows, most likely the Windows Credential Manager is full. To make room in the Windows Credential Manager

1. Close Storage Explorer
2. On the **Start** menu, search for **Credential Manager** and open it.
3. Go to **Windows Credentials**.
4. Under **Generic Credentials**, look for entries associated with programs you no longer use and delete them.  
You can also look for entries like `azcopy/aadtoken/<some number>` and delete those.

If the message continues to appear after completing the above steps, or if you encounter this message on platforms other than Windows, then please [open an issue on GitHub](#).

## Can't remove an attached storage account or resource

If you can't remove an attached account or storage resource through the UI, you can manually delete all attached resources by deleting the following folders:

- Windows: `%AppData%/StorageExplorer`
- macOS: `/Users/<your_name>/Library/Application Support/StorageExplorer`
- Linux: `~/.config/StorageExplorer`

Close Storage Explorer before you delete these folders.

### NOTE

If you've ever imported any SSL certificates, back up the contents of the `certs` directory. Later, you can use the backup to reimport your SSL certificates.

## Proxy issues

Storage Explorer supports connecting to Azure Storage resources via a proxy server. If you experience any issues when you connect to Azure via proxy, here are some suggestions.

Storage Explorer only supports basic authentication with proxy servers. Other authentication methods, such as NTLM, aren't supported.

### NOTE

Storage Explorer doesn't support proxy autoconfig files for configuring proxy settings.

## Verify Storage Explorer proxy settings

The Application > Proxy > Proxy configuration setting determines which source Storage Explorer gets the proxy configuration from.

If you select Use environment variables, make sure to set the `HTTPS_PROXY` or `HTTP_PROXY` environment variables. Environment variables are case sensitive, so be sure to set the correct variables. If these variables are undefined or invalid, Storage Explorer won't use a proxy. Restart Storage Explorer after you modify any environment variables.

If you select Use app proxy settings, make sure the in-app proxy settings are correct.

## Steps for diagnosing issues

If you're still experiencing issues, try these troubleshooting methods:

1. If you can connect to the internet without using your proxy, verify that Storage Explorer works without proxy settings enabled. If Storage Explorer connects successfully, there might be an issue with your proxy server. Work with your admin to identify the problems.
2. Verify that other applications that use the proxy server work as expected.
3. Verify that you can connect to the portal for the Azure environment you're trying to use.
4. Verify that you can receive responses from your service endpoints. Enter one of your endpoint URLs into your browser. If you can connect, you should receive an `InvalidParameterValue` or similar XML response.
5. Check whether someone else using Storage Explorer with the same proxy server can connect. If they can, you might have to contact your proxy server admin.

## Tools for diagnosing issues

A networking tool, such as Fiddler, can help you diagnose problems.

1. Configure your networking tool as a proxy server running on the local host. If you have to continue working behind an actual proxy, you might have to configure your networking tool to connect through the proxy.
2. Check the port number used by your networking tool.
3. Configure Storage Explorer proxy settings to use the local host and the networking tool's port number, such as "localhost:8888".

When set correctly, your networking tool will log network requests made by Storage Explorer to management and service endpoints.

If your networking tool doesn't appear to be logging Storage Explorer traffic, try testing your tool with a different application. For example, enter the endpoint URL for one of your storage resources, such as

`https://contoso.blob.core.windows.net/`) in a web browser. You'll receive a response similar to this code sample.

```
<?xml version="1.0" encoding="UTF-8"?>
- <Error>
 <Code>InvalidParameterValue</Code>
 <Message>Value for one of the query parameters specified in the request URI is invalid.
RequestId:57d9d9e5-0001-0008-0143-b28062000000 Time:2017-04-
 10T21:42:17.3863214Z</Message>
 <QueryParameterName>comp</QueryParameterName>
 <QueryParameterValue/>
 <Reason/>
</Error>
```

The response suggests the resource exists, even though you can't access it.

If your networking tool only shows traffic from other applications, you might need to adjust the proxy settings in Storage Explorer. Otherwise, you might need to adjust your tool's settings.

## Contact proxy server admin

If your proxy settings are correct, you might have to contact your proxy server admin to:

- Make sure your proxy doesn't block traffic to Azure management or resource endpoints.
- Verify the authentication protocol used by your proxy server. Storage Explorer only supports basic authentication protocols. Storage Explorer doesn't support NTLM proxies.

## "Unable to Retrieve Children" error message

If you're connected to Azure through a proxy, verify that your proxy settings are correct.

If the owner of a subscription or account has granted you access to a resource, verify that you have read or list permissions for that resource.

## Connection string doesn't have complete configuration settings

If you receive this error message, it's possible that you don't have the necessary permissions to obtain the keys for your storage account. To confirm that this is the case, go to the portal and locate your storage account. Right-click the node for your storage account and select **Open in Portal**. Then, go to the **Access Keys** pane. If you don't have permissions to view keys, you'll see a "You don't have access" message. To work around this issue, you can either obtain the account key from someone else and attach through the name and key or you can ask someone for a shared access signature to the storage account and use it to attach the storage account.

If you do see the account keys, file an issue in GitHub so that we can help you resolve the issue.

## "Error occurred while adding new connection: TypeError: Cannot read property 'version' of undefined"

If you receive this error message when you try to add a custom connection, the connection data that's stored in the local credential manager might be corrupted. To work around this issue, try deleting your corrupted local connections, and then re-add them:

1. Start Storage Explorer. From the menu, go to **Help > Toggle Developer Tools**.
2. In the opened window, on the **Application** tab, go to **Local Storage > file://** on the left side.
3. Depending on the type of connection you're having an issue with, look for its key. Then copy its value into a text editor. The value is an array of your custom connection names, such as:
  - Storage accounts
    - `StorageExplorer_CustomConnections_Accounts_v1`
  - Blob containers
    - `StorageExplorer_CustomConnections_Blobs_v1`
    - `StorageExplorer_CustomConnections_Blobs_v2`
  - File shares
    - `StorageExplorer_CustomConnections_Files_v1`
  - Queues
    - `StorageExplorer_CustomConnections_Queue_v1`
  - Tables
    - `StorageExplorer_CustomConnections_Tables_v1`
4. After you save your current connection names, set the value in **Developer Tools** to `[]`.

To preserve the connections that aren't corrupted, use the following steps to locate the corrupted connections. If you don't mind losing all existing connections, skip these steps and follow the platform-specific instructions to clear your connection data.

1. From a text editor, re-add each connection name to **Developer Tools**. Then check whether the connection is still working.
2. If a connection is working correctly, it's not corrupted and you can safely leave it there. If a connection isn't working, remove its value from **Developer Tools**, and record it so that you can add it back later.
3. Repeat until you've examined all your connections.

After you go through all your connections, for all connection names that aren't added back, you must clear their corrupted data, if there is any. Then add them back by using the standard steps in Storage Explorer.

- [Windows](#)
- [macOS](#)
- [Linux](#)

1. On the **Start** menu, search for **Credential Manager** and open it.
2. Go to **Windows Credentials**.
3. Under **Generic Credentials**, look for entries that have the `<connection_type_key>/<corrupted_connection_name>` key. An example is `StorageExplorer_CustomConnections_Accounts_v1/account1`.
4. Delete these entries and re-add the connections.

If you still encounter this error after you run these steps, or if you want to share what you suspect has corrupted the connections, [open an issue](#) on our GitHub page.

## Issues with a shared access signature URL

If you connect to a service through a shared access signature URL and experience an error:

- Verify that the URL provides the necessary permissions to read or list resources.
- Verify that the URL hasn't expired.
- If the shared access signature URL is based on an access policy, verify that the access policy hasn't been revoked.

If you accidentally attached by using an invalid shared access signature URL and now can't detach, follow these steps:

1. When you're running Storage Explorer, select **F12** to open the **Developer Tools** window.
2. On the **Application** tab, select **Local Storage** > `file://` on the left side.
3. Find the key associated with the service type of the problematic shared access signature URI. For example, if the bad shared access signature URI is for a blob container, look for the key named `StorageExplorer_AddStorageServiceSAS_v1_blob`.
4. The value of the key should be a JSON array. Find the object associated with the bad URI, and delete it.
5. Select **Ctrl+R** to reload Storage Explorer.

## Linux dependencies

### Snap

Storage Explorer 1.10.0 and later is available as a snap from the Snap Store. The Storage Explorer snap installs all its dependencies automatically. It's updated when a new version of the snap is available. Installing the Storage Explorer snap is the recommended method of installation.

Storage Explorer requires the use of a password manager, which you might need to connect manually before Storage Explorer will work correctly. You can connect Storage Explorer to your system's password manager by running the following command:

```
snap connect storage-explorer:password-manager-service :password-manager-service
```

### .tar.gz file

You can also download the application as a *.tar.gz* file, but you'll have to install dependencies manually.

Storage Explorer as provided in the *.tar.gz* download is supported for the following versions of Ubuntu only.

Storage Explorer might work on other Linux distributions, but they aren't officially supported.

- Ubuntu 20.04 x64
- Ubuntu 18.04 x64
- Ubuntu 16.04 x64

Storage Explorer requires the .NET 6 runtime to be installed on your system. The ASP.NET runtime is **not** required.

#### NOTE

Older versions of Storage Explorer may require a different version of .NET or .NET Core. Refer to release notes or in app error messages to help determine the required version.

- [Ubuntu 22.04](#)
- [Ubuntu 20.04](#)
- [Ubuntu 18.04](#)

1. Download the Storage Explorer *.tar.gz* file.

2. Install the [.NET 6 runtime](#)

Many libraries needed by Storage Explorer come preinstalled with Canonical's standard installations of Ubuntu.

Custom environments might be missing some of these libraries. If you have issues launching Storage Explorer, make sure the following packages are installed on your system:

- iproute2
- libasound2
- libatm1
- libgconf-2-4
- libnspr4
- libnss3
- libpulse0
- libsecret-1-0
- libx11-xcb1
- libxss1
- libxtables11
- libxtst6
- xdg-utils

### Patch Storage Explorer for newer versions of .NET Core

For Storage Explorer 1.7.0 or earlier, you might have to patch the version of .NET Core used by Storage Explorer:

1. Download version 1.5.43 of StreamJsonRpc [from NuGet](#).1. Look for the **Download package** link on the right side of the page.
2. After you download the package, change its file extension from *.nupkg* to *.zip*.

3. Unzip the package.
4. Open the `streamjsonrpc.1.5.43/lib/netstandard1.1/` folder.
5. Copy `StreamJsonRpc.dll` to the following locations in the Storage Explorer folder:
  - `StorageExplorer/resources/app/ServiceHub/Services/Microsoft.Developer.IdentityService/`
  - `StorageExplorer/resources/app/ServiceHub/Hosts/ServiceHub.Host.Core.CLR.x64/`

## Open In Explorer button in the Azure portal doesn't work

If the **Open In Explorer** button in the Azure portal doesn't work, make sure you're using a compatible browser. The following browsers were tested for compatibility:

- Microsoft Edge
- Mozilla Firefox
- Google Chrome
- Microsoft Internet Explorer

## Gather logs

When you report an issue to GitHub, you might be asked to gather certain logs to help diagnose your issue.

### Storage Explorer logs

Starting with version 1.16.0, Storage Explorer logs various things to its own application logs. You can easily get to these logs by selecting **Help > Open Logs Directory**. By default, Storage Explorer logs at a low level of verbosity. To change the verbosity level, add an environment variable with the name of `STG_EX_LOG_LEVEL`, and any of the following values:

- `silent`
- `critical`
- `error`
- `warning`
- `info` (default level)
- `verbose`
- `debug`

Logs are split into folders for each session of Storage Explorer that you run. For whatever log files you need to share, place them in a zip archive, with files from different sessions in different folders.

### Authentication logs

For issues related to sign-in or Storage Explorer's authentication library, you'll most likely need to gather authentication logs. Authentication logs are stored at:

- Windows: `C:\Users\<your username>\AppData\Local\Temp\servicehub\logs`
- macOS and Linux: `~/ServiceHub/logs`

Generally, you can follow these steps to gather the logs:

1. Go to **Settings** (the **gear** symbol on the left) > **Application** > **Sign-in**. Select **Verbose Authentication Logging**. If Storage Explorer fails to start because of an issue with its authentication library, this will be done for you.
2. Close Storage Explorer.
3. Optional/recommended: Clear out existing logs from the `logs` folder. This step reduces the amount of information you have to send us.

4. Open Storage Explorer and reproduce your issue.
5. Close Storage Explorer.
6. Zip the contents of the `/logs` folder.

## AzCopy logs

If you're having trouble transferring data, you might need to get the AzCopy logs. AzCopy logs can be found easily via two different methods:

- For failed transfers still in the Activity Log, select [Go to AzCopy Log File](#).
- For transfers that failed in the past, go to the AzCopy logs folder. This folder can be found at:
  - Windows: `C:\Users\<your username>\.azcopy`
  - macOS and Linux: `~/.azcopy`

## Network logs

For some issues, you'll need to provide logs of the network calls made by Storage Explorer. On Windows, you can do this step by using Fiddler.

### NOTE

Fiddler traces might contain passwords you entered or sent in your browser during the gathering of the trace. Make sure to read the instructions on how to sanitize a Fiddler trace. Don't upload Fiddler traces to GitHub. You'll be told where you can securely send your Fiddler trace.

### Part 1: Install and configure Fiddler

1. Install Fiddler.
2. Start Fiddler.
3. Go to **Tools > Options**.
4. Select the **HTTPS** tab.
5. Make sure **Capture CONNECTs** and **Decrypt HTTPS traffic** are selected.
6. Select **Actions**.
7. Select **Trust Root Certificate** and then select **Yes** in the next dialog.
8. Select **Actions** again.
9. Select **Export Root Certificate to Desktop**.
10. Go to your desktop, find the `FiddlerRoot.cer` file, and double-click it.
11. Go to the **Details** tab.
12. Select **Copy to File**.
13. In the export wizard, choose the following options:
  - Base-64 encoded X.509.
  - For file name, browse to `C:\Users\<your user dir>\AppData\Roaming\StorageExplorer\certs`. Then you can save it as any file name.
14. Close the certificate window.
15. Start Storage Explorer.
16. Go to **Edit > Configure Proxy**.

17. In the dialog, select **Use app proxy settings**. Set the URL to `http://localhost` and the port to **8888**.

18. Select **OK**.

19. Restart Storage Explorer.

20. You should start seeing network calls from a `storageexplorer:` process show up in Fiddler.

#### Part 2: Reproduce the issue

1. Close all apps other than Fiddler.
2. Clear the Fiddler log by using the X in the top left, near the **View** menu.
3. Optional/recommended: Let Fiddler set for a few minutes. If you see network calls appear that aren't related to Storage Explorer, right-click them and select **Filter Now > Hide (process name)**.
4. Start Storage Explorer.
5. Reproduce the issue.
6. Select **File > Save > All Sessions**. Save it somewhere you won't forget.
7. Close Fiddler and Storage Explorer.

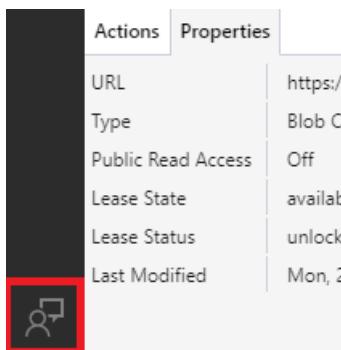
#### Part 3: Sanitize the Fiddler trace

1. Double-click the Fiddler trace (.saz file).
2. Select **Ctrl+F**.
3. In the dialog that appears, make sure the following options are set: **Search = Requests and responses** and **Examine = Headers and bodies**.
4. Search for any passwords you used while you collected the Fiddler trace and any entries that are highlighted. Right-click and select **Remove > Selected sessions**.
5. If you definitely entered passwords into your browser while you collected the trace but you don't find any entries when you use **Ctrl+F**, you don't want to change your passwords, or if the passwords you used are used for other accounts, skip sending us the .saz file.
6. Save the trace again with a new name.
7. Optional: Delete the original trace.

## Next steps

If none of these solutions work for you, you can:

- [Create a support ticket](#).
- [Open an issue on GitHub](#) by selecting the **Report issue to GitHub** button in the lower-left corner.



# Azure Storage Explorer security guide

8/22/2022 • 4 minutes to read • [Edit Online](#)

Microsoft Azure Storage Explorer enables you to easily work with Azure Storage data safely and securely on Windows, macOS, and Linux. By following these guidelines, you can ensure your data stays protected.

## General

- **Always use the latest version of Storage Explorer.** Storage Explorer releases may contain security updates. Staying up to date helps ensure general security.
- **Only connect to resources you trust.** Data that you download from untrusted sources could be malicious, and uploading data to an untrusted source may result in lost or stolen data.
- **Use HTTPS whenever possible.** Storage Explorer uses HTTPS by default. Some scenarios allow you to use HTTP, but HTTP should be used only as a last resort.
- **Ensure only the needed permissions are given to the people who need them.** Avoid being overly permissive when granting anyone access to your resources.
- **Use caution when executing critical operations.** Certain operations, such as delete and overwrite, are irreversible and may cause data loss. Make sure you're working with the correct resources before executing these operations.

## Choosing the right authentication method

Storage Explorer provides various ways to access your Azure Storage resources. Whatever method you choose, here are our recommendations.

### Azure AD authentication

The easiest and most secure way to access your Azure Storage resources is to sign in with your Azure account. Signing in uses Azure AD authentication, which allows you to:

- Give access to specific users and groups.
- Revoke access to specific users and groups at any time.
- Enforce access conditions, such as requiring multi-factor authentication.

We recommend using Azure AD authentication whenever possible.

This section describes the two Azure AD-based technologies that can be used to secure your storage resources.

#### Azure role-based access control (Azure RBAC)

[Azure role-based access control \(Azure RBAC\)](#) give you fine-grained access control over your Azure resources. Azure roles and permissions can be managed from the Azure portal.

Storage Explorer supports Azure RBAC access to Storage Accounts, Blobs, and Queues. If you need access to File Shares or Tables, you'll need to assign Azure roles that grant permission to list storage account keys.

#### Access control lists (ACLs)

[Access control lists \(ACLs\)](#) let you control file and folder level access in ADLS Gen2 blob containers. You can manage your ACLs using Storage Explorer.

#### Shared access signatures (SAS)

If you can't use Azure AD authentication, we recommend using shared access signatures. With shared access signatures, you can:

- Provide anonymous limited access to secure resources.
- Revoke a SAS immediately if generated from a shared access policy (SAP).

However, with shared access signatures, you can't:

- Restrict who can use a SAS. A valid SAS can be used by anyone who has it.
- Revoke a SAS if not generated from a shared access policy (SAP).

When using SAS in Storage Explorer, we recommend the following guidelines:

- **Limit the distribution of SAS tokens and URIs.** Only distribute SAS tokens and URIs to trusted individuals. Limiting SAS distribution reduces the chance a SAS could be misused.
- **Only use SAS tokens and URIs from entities you trust.**
- **Use shared access policies (SAP) when generating SAS tokens and URIs if possible.** A SAS based on a shared access policy is more secure than a bare SAS, because the SAS can be revoked by deleting the SAP.
- **Generate tokens with minimal resource access and permissions.** Minimal permissions limit the potential damage that could be done if a SAS is misused.
- **Generate tokens that are only valid for as long as necessary.** A short lifespan is especially important for bare SAS, because there's no way to revoke them once generated.

#### IMPORTANT

When sharing SAS tokens and URIs for troubleshooting purposes, such as in service requests or bug reports, always redact at least the signature portion of the SAS.

## Storage account keys

Storage account keys grant unrestricted access to the services and resources within a storage account. For this reason, we recommend limiting the use of keys to access resources in Storage Explorer. Use Azure RBAC features or SAS to provide access instead.

Some Azure roles grant permission to retrieve storage account keys. Individuals with these roles can effectively circumvent permissions granted or denied by Azure RBAC. We recommend not granting this permission unless it's necessary.

Storage Explorer will attempt to use storage account keys, if available, to authenticate requests. You can disable this feature in Settings (**Services > Storage Accounts > Disable Usage of Keys**). Some features don't support Azure RBAC, such as working with Classic storage accounts. Such features still require keys and are not affected by this setting.

If you must use keys to access your storage resources, we recommend the following guidelines:

- **Don't share your account keys with anyone.**
- **Treat your storage account keys like passwords.** If you must make your keys accessible, use secure storage solutions such as [Azure Key Vault](#).

#### NOTE

If you believe a storage account key has been shared or distributed by mistake, you can generate new keys for your storage account from the Azure portal.

## Public access to blob containers

Storage Explorer allows you to modify the access level of your Azure Blob Storage containers. Non-private blob containers allow anyone anonymous read access to data in those containers.

When enabling public access for a blob container, we recommend the following guidelines:

- **Don't enable public access to a blob container that may contain any potentially sensitive data.**  
Make sure your blob container is free of all private data.
- **Don't upload any potentially sensitive data to a blob container with Blob or Container access.**

## Next steps

- [Security recommendations](#)

# Azure Storage Explorer blob versioning guide

8/22/2022 • 3 minutes to read • [Edit Online](#)

Microsoft Azure Storage Explorer provides easy access and management of blob versions. This guide will help you understand how blob versioning works in Storage Explorer. Before continuing, it's recommended you read more about [blob versioning](#).

## Terminology

This section provides some definitions to help understand their usage in this article.

- Soft delete: An alternative automatic data protection feature. You can learn more about soft delete [here](#).
- Active blob: A blob or blob version is created in active state. You can only operate on blobs or blob versions in active state.
- Soft-deleted blob: A blob or blob version marked as soft-deleted. Soft-deleted blobs are only kept for its retention period.
- Blob version: A blob created with blob versioning enabled. Each blob version is associated with a version ID.
- Current version: A blob version marked as the current version.
- Previous version: A blob version that isn't the current version.
- Non-version blob: A blob created with blob versioning disabled. A non-version blob doesn't have a version ID.

## View blob versions

Storage Explorer supports four different views for viewing blobs.

VIEW	ACTIVE NON-VERSION BLOBS	SOFT-DELETED NON-VERSION BLOBS	BLOB VERSIONS
Active blobs	Yes	No	Current version only
Active blobs and soft-deleted blobs	Yes	Yes	Current version only
Active blobs and blobs without current version	Yes	No	Current version or latest active version
All blobs and blobs without current version	Yes	Yes	Current version or latest version

### Active blobs

In this view, Storage Explorer displays:

- Active non-version blobs
- Current versions

### Active blobs and soft-deleted blobs

In this view, Storage Explorer displays:

- Active non-version blobs

- Soft-deleted non-version blobs
- Current versions.

### Active blobs and blobs without current version

In this view, Storage Explorer displays:

- Active non-version blobs
- Current versions
- Latest active previous versions.

For blobs that don't have a current version but have an active previous version, Storage Explorer displays their latest active previous version as a representation of that blob.

### All blobs and blobs without current version

In this view, Storage Explorer displays:

- Active non-version blobs
- Soft-deleted non-version blobs
- Current versions
- Latest previous versions.

For blobs that don't have a current version, Storage Explorer displays their latest previous version as a representation of that blob.

#### NOTE

Due to service limitation, Storage Explorer needs some additional processing to get a hierarchical view of your virtual directories when listing blob versions. It will take longer to list blobs in the following views:

- Active blobs and blobs without current version
- All blobs and blobs without current version

## Manage blob versions

### View versions of a blob

Storage Explorer provides a **Manage Versions** command to view all the versions of a blob. To view a blob's versions, select the blob you want to view versions for and select **Manage History** → **Manage Versions** from either the toolbar or the context menu.

### Download blob versions

To download one or more blob versions, select the blob versions you want to download and select **Download** from the toolbar or the context menu.

If you're downloading multiple versions of a blob, the downloaded files will have their version IDs at the beginning of their file names.

### Delete blob versions

To delete one or more blob versions, select the blob versions you want to delete and select **Delete** from the toolbar or the context menu.

Blob versions are subject to your soft-delete policy. If soft-delete is enabled, blob versions will be soft-deleted. One special case is deleting a current version. Deleting a current version will automatically make it become an active previous version instead.

### Promote blob version

You can restore the contents of a blob by promoting a previous version to become the current version. Select the blob version you want to promote and select **Promote Version** from the toolbar or the context menu.

Non-version blobs will be overwritten by the promoted blob version. Make sure you no longer need that data or back up the data yourself before confirming the operation. Current versions automatically become previous versions, so Storage Explorer won't prompt for confirmation.

### **Undelete blob version**

Blob versions can't be undeleted individually. They must be undeleted all at once. To undelete all blob versions of a blob, select any one of the blob's versions and select **Undelete Selected** from the toolbar or the context menu.

### **Change access tier of blob versions**

Each blob version has its own access tier. To change access tier of blob versions, select the blob versions you want to change access tier and select **Change Access Tier...** from the context menu.

## See Also

- [Blob versioning](#)
- [Soft delete for blobs](#)
- [Azure Storage Explorer soft delete guide](#)

# Manage Azure Blob Storage resources with Storage Explorer

8/22/2022 • 7 minutes to read • [Edit Online](#)

## Overview

[Azure Blob Storage](#) is a service for storing large amounts of unstructured data, such as text or binary data, that can be accessed from anywhere in the world via HTTP or HTTPS. You can use Blob storage to expose data publicly to the world, or to store application data privately. In this article, you'll learn how to use Storage Explorer to work with blob containers and blobs.

## Prerequisites

To complete the steps in this article, you'll need the following:

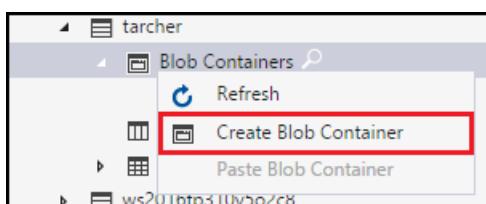
- [Download and install Storage Explorer](#)
- [Connect to an Azure storage account or service](#)

## Create a blob container

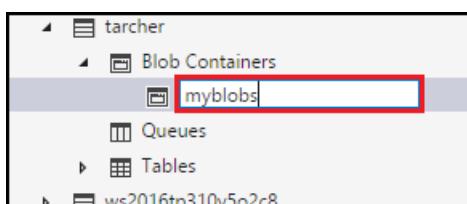
All blobs must reside in a blob container, which is simply a logical grouping of blobs. An account can contain an unlimited number of containers, and each container can store an unlimited number of blobs.

The following steps illustrate how to create a blob container within Storage Explorer.

1. Open Storage Explorer.
2. In the left pane, expand the storage account within which you wish to create the blob container.
3. Right-click **Blob Containers**, and - from the context menu - select **Create Blob Container**.



4. A text box will appear below the **Blob Containers** folder. Enter the name for your blob container. See [Create a container](#) for information on rules and restrictions on naming blob containers.



5. Press **Enter** when done to create the blob container, or **Esc** to cancel. Once the blob container has been successfully created, it will be displayed under the **Blob Containers** folder for the selected storage account.

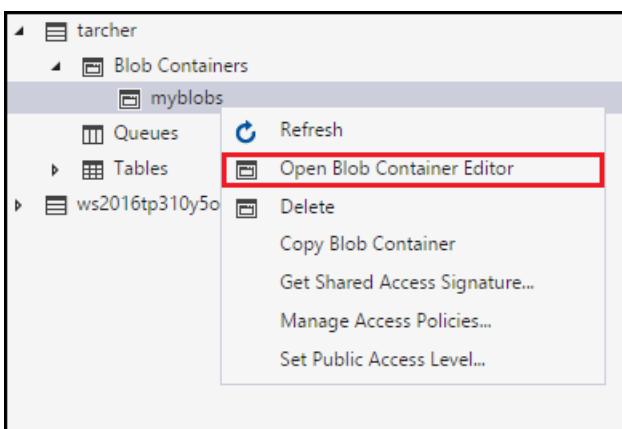


## View a blob container's contents

Blob containers contain blobs and folders (that can also contain blobs).

The following steps illustrate how to view the contents of a blob container within Storage Explorer:

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the blob container you wish to view.
3. Expand the storage account's **Blob Containers**.
4. Right-click the blob container you wish to view, and - from the context menu - select **Open Blob Container Editor**. You can also double-click the blob container you wish to view.



5. The main pane will display the blob container's contents.

A screenshot of the Storage Explorer main pane. The top navigation bar includes buttons for Upload, Download, Open, Copy URL, Select all, Copy, Paste, Delete, and Refresh. Below the navigation bar is a search bar with the text 'myblobs' and a 'Search by prefix' button. A table displays the contents of the 'myblobs' container:

Name	Last Modified	Blob Type	Content Type	Size
reachin.png	Tue, 17 May 2016 20:21:44 GMT	Block Blob	image/png	437.8 KB

Showing 1 to 1 of 1 loaded items.

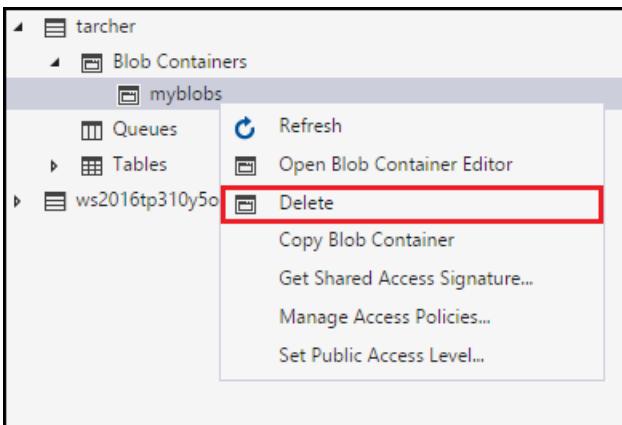
## Delete a blob container

Blob containers can be easily created and deleted as needed. (To see how to delete individual blobs, refer to the section, [Managing blobs in a blob container](#).)

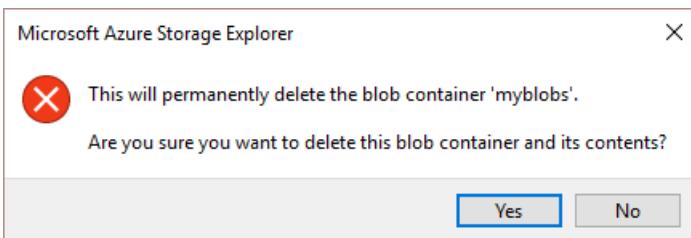
The following steps illustrate how to delete a blob container within Storage Explorer:

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the blob container you wish to view.

3. Expand the storage account's **Blob Containers**.
4. Right-click the blob container you wish to delete, and - from the context menu - select **Delete**. You can also press **Delete** to delete the currently selected blob container.



5. Select **Yes** to the confirmation dialog.

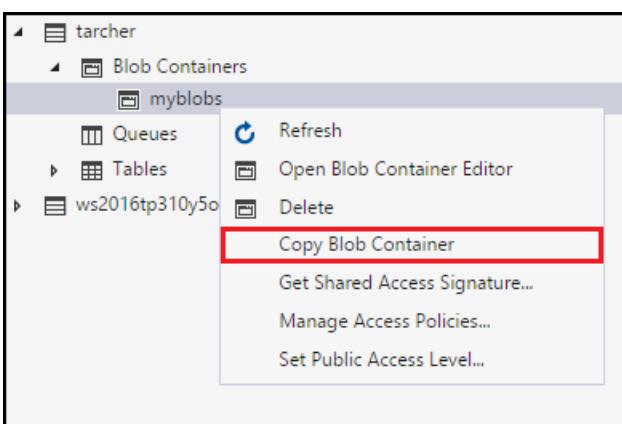


## Copy a blob container

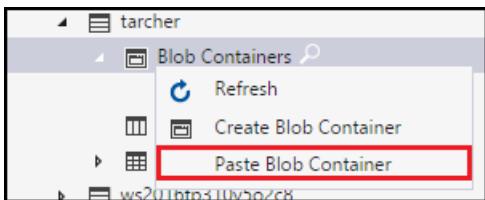
Storage Explorer enables you to copy a blob container to the clipboard, and then paste that blob container into another storage account. (To see how to copy individual blobs, refer to the section, [Managing blobs in a blob container](#).)

The following steps illustrate how to copy a blob container from one storage account to another.

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the blob container you wish to copy.
3. Expand the storage account's **Blob Containers**.
4. Right-click the blob container you wish to copy, and - from the context menu - select **Copy Blob Container**.



5. Right-click the desired "target" storage account into which you want to paste the blob container, and - from the context menu - select **Paste Blob Container**.

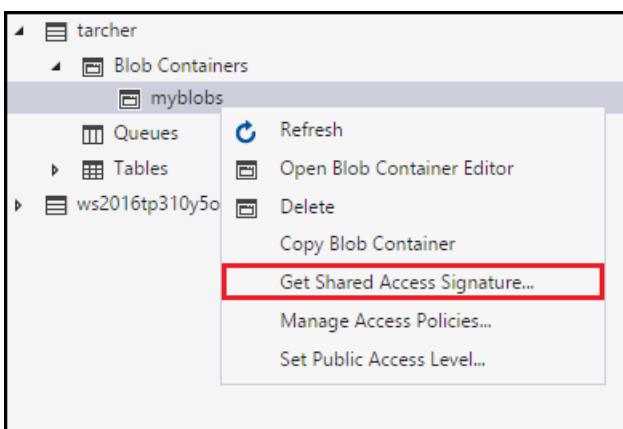


## Get the SAS for a blob container

A [shared access signature \(SAS\)](#) provides delegated access to resources in your storage account. This means that you can grant a client limited permissions to objects in your storage account for a specified period of time and with a specified set of permissions, without having to share your account access keys.

The following steps illustrate how to create a SAS for a blob container:

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the blob container for which you wish to get a SAS.
3. Expand the storage account's **Blob Containers**.
4. Right-click the desired blob container, and - from the context menu - select **Get Shared Access Signature**.



5. In the **Shared Access Signature** dialog, specify the policy, start and expiration dates, time zone, and access levels you want for the resource.

### Shared Access Signature

Access policy:

Start time:

Expiry time:

Time zone:  
 Local  
 UTC

Permissions:

Read  
 Write  
 Delete  
 List

6. When you're finished specifying the SAS options, select **Create**.
7. A second **Shared Access Signature** dialog will then display that lists the blob container along with the URL and QueryStrings you can use to access the storage resource. Select **Copy** next to the URL you wish to copy to the clipboard.

### Shared Access Signature

Container:

URL:

Query string:

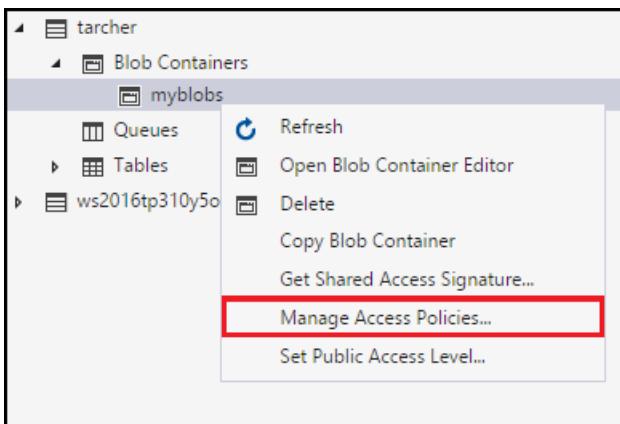
8. When done, select **Close**.

## Manage Access Policies for a blob container

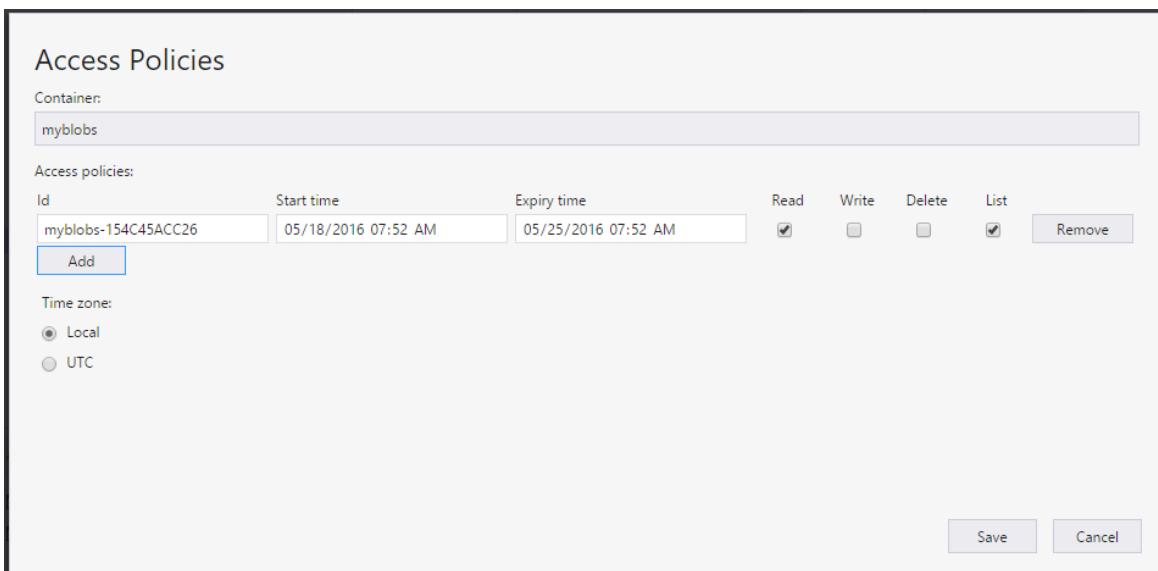
The following steps illustrate how to manage (add and remove) access policies for a blob container:

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the blob container whose access policies you wish to manage.
3. Expand the storage account's **Blob Containers**.

4. Select the desired blob container, and - from the context menu - select **Manage Access Policies**.



5. The **Access Policies** dialog will list any access policies already created for the selected blob container.



6. Follow these steps depending on the access policy management task:

- **Add a new access policy** - Select Add. Once generated, the **Access Policies** dialog will display the newly added access policy (with default settings).
- **Edit an access policy** - Make any desired edits, and select Save.
- **Remove an access policy** - Select Remove next to the access policy you wish to remove.

#### NOTE

Modifying immutability policies is not supported from Storage Explorer.

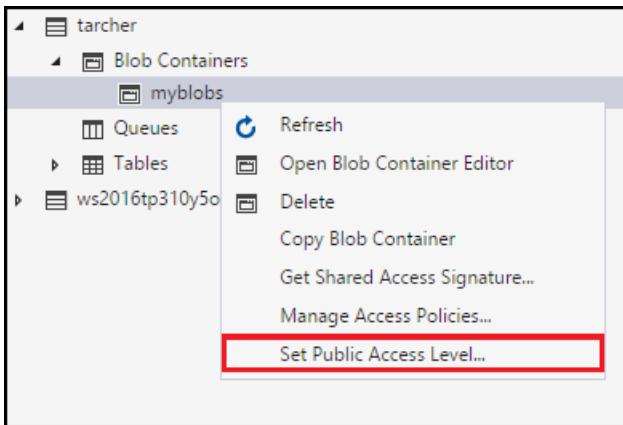
## Set the Public Access Level for a blob container

By default, every blob container is set to "No public access".

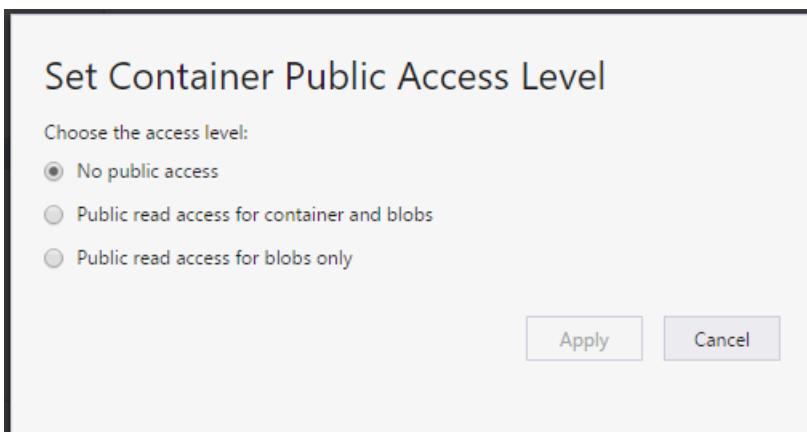
The following steps illustrate how to specify a public access level for a blob container.

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the blob container whose access policies you wish to manage.
3. Expand the storage account's **Blob Containers**.

4. Select the desired blob container, and - from the context menu - select Set Public Access Level.



5. In the Set Container Public Access Level dialog, specify the desired access level.



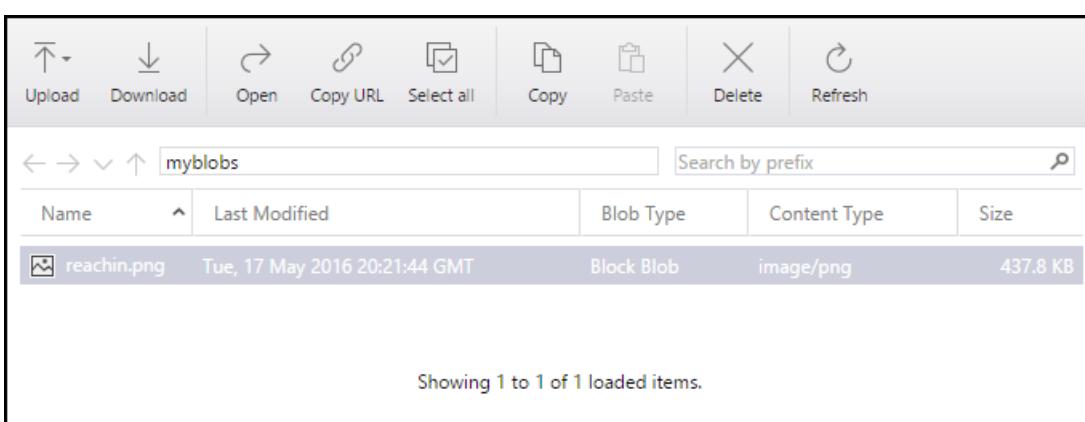
6. Select Apply.

## Managing blobs in a blob container

Once you've created a blob container, you can upload a blob to that blob container, download a blob to your local computer, open a blob on your local computer, and much more.

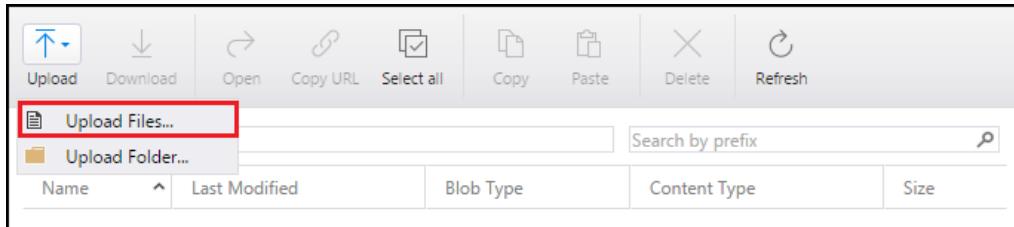
The following steps illustrate how to manage the blobs (and folders) within a blob container.

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the blob container you wish to manage.
3. Expand the storage account's **Blob Containers**.
4. Double-click the blob container you wish to view.
5. The main pane will display the blob container's contents.

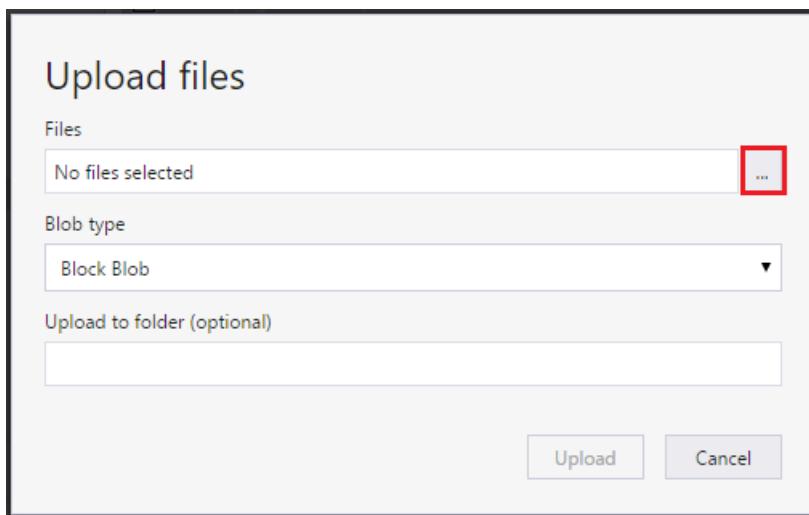


6. The main pane will display the blob container's contents.
7. Follow these steps depending on the task you wish to perform:
  - **Upload files to a blob container**

- a. On the main pane's toolbar, select **Upload**, and then **Upload Files** from the drop-down menu.



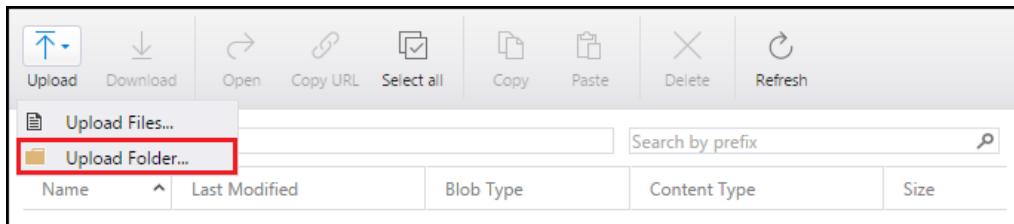
- b. In the **Upload files** dialog, select the ellipsis (...) button on the right side of the **Files** text box to select the file(s) you wish to upload.



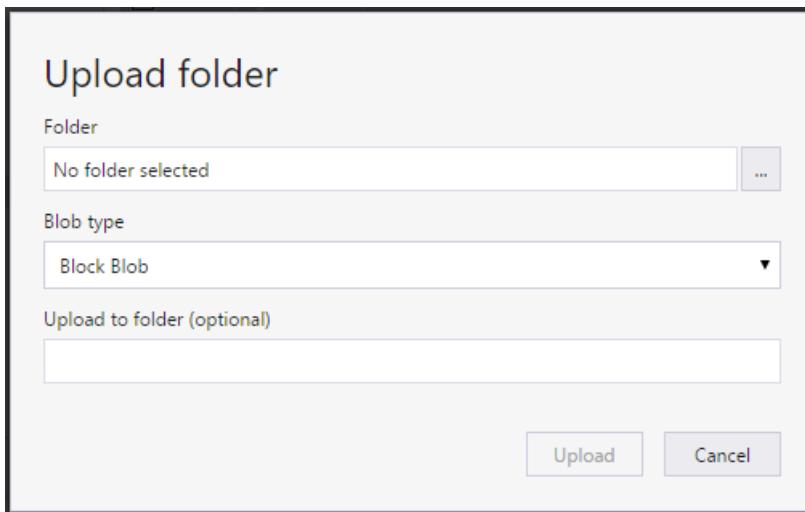
- c. Specify the type of **Blob type**. See [Create a container](#) for more information.
  - d. Optionally, specify a target folder into which the selected file(s) will be uploaded. If the target folder doesn't exist, it will be created.
  - e. Select **Upload**.

- **Upload a folder to a blob container**

- a. On the main pane's toolbar, select **Upload**, and then **Upload Folder** from the drop-down menu.



- b. In the **Upload folder** dialog, select the ellipsis (...) button on the right side of the **Folder** text box to select the folder whose contents you wish to upload.



- c. Specify the type of **Blob type**. See [Create a container](#) for more information.
  - d. Optionally, specify a target folder into which the selected folder's contents will be uploaded.  
If the target folder doesn't exist, it will be created.
  - e. Select **Upload**.
- **Download a blob to your local computer**
    - a. Select the blob you wish to download.
    - b. On the main pane's toolbar, select **Download**.
    - c. In the **Specify where to save the downloaded blob** dialog, specify the location where you want the blob downloaded, and the name you wish to give it.
    - d. Select **Save**.
  - **Open a blob on your local computer**
    - a. Select the blob you wish to open.
    - b. On the main pane's toolbar, select **Open**.
    - c. The blob will be downloaded and opened using the application associated with the blob's underlying file type.
  - **Copy a blob to the clipboard**
    - a. Select the blob you wish to copy.
    - b. On the main pane's toolbar, select **Copy**.
    - c. In the left pane, navigate to another blob container, and double-click it to view it in the main pane.
    - d. On the main pane's toolbar, select **Paste** to create a copy of the blob.
  - **Delete a blob**
    - a. Select the blob you wish to delete.
    - b. On the main pane's toolbar, select **Delete**.
    - c. Select **Yes** to the confirmation dialog.

## Next steps

- View the [latest Storage Explorer release notes and videos](#).
- Learn how to [create applications using Azure blobs, tables, queues, and files](#).

# Using Storage Explorer with Azure Files

8/22/2022 • 7 minutes to read • [Edit Online](#)

Azure Files is a service that offers file shares in the cloud using the standard Server Message Block (SMB) Protocol. Both SMB 2.1 and SMB 3.0 are supported. With Azure Files, you can migrate legacy applications that rely on file shares to Azure quickly and without costly rewrites. You can use File storage to expose data publicly to the world, or to store application data privately. In this article, you'll learn how to use Storage Explorer to work with file shares and files.

## Prerequisites

To complete the steps in this article, you'll need the following:

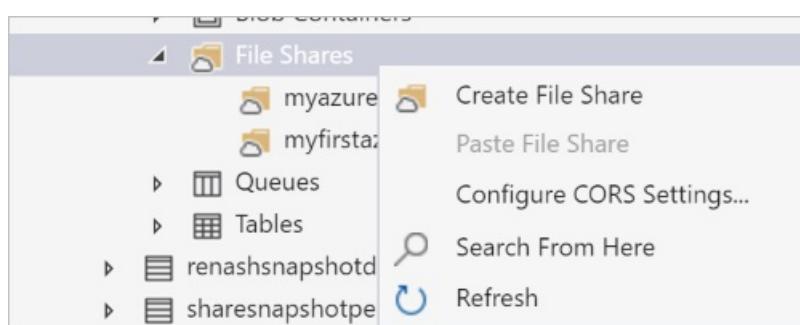
- [Download and install Storage Explorer](#)
- [Connect to an Azure storage account or service](#)

## Create a file share

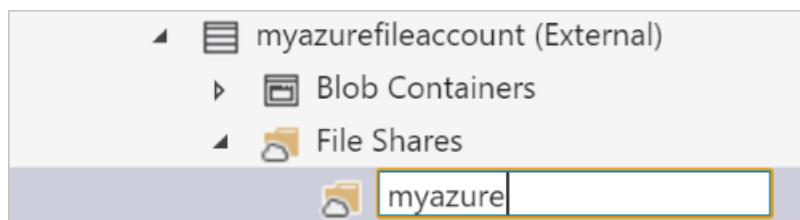
All files must reside in a file share, which is simply a logical grouping of files. An account can contain an unlimited number of file shares, and each share can store an unlimited number of files.

The following steps illustrate how to create a file share within Storage Explorer.

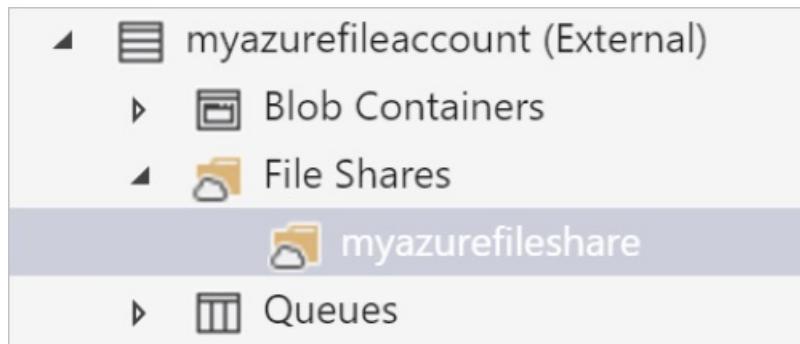
1. Open Storage Explorer.
2. In the left pane, expand the storage account within which you wish to create the file share
3. Right-click **File Shares**, and - from the context menu - select **Create File Share**.



4. A text box will appear below the **File Shares** folder. Enter the name for your file share. See the [Share naming rules](#) section for a list of rules and restrictions on naming file shares.



5. Press **Enter** when done to create the file share, or **Esc** to cancel. Once the file share has been successfully created, it will be displayed under the **File Shares** folder for the selected storage account.

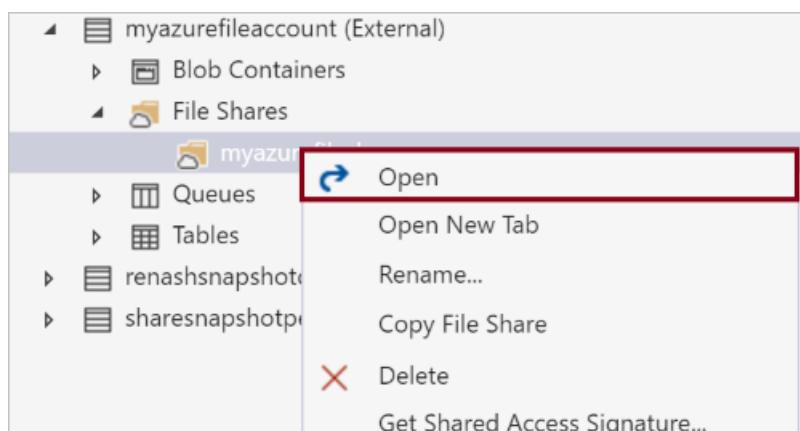


## View a file share's contents

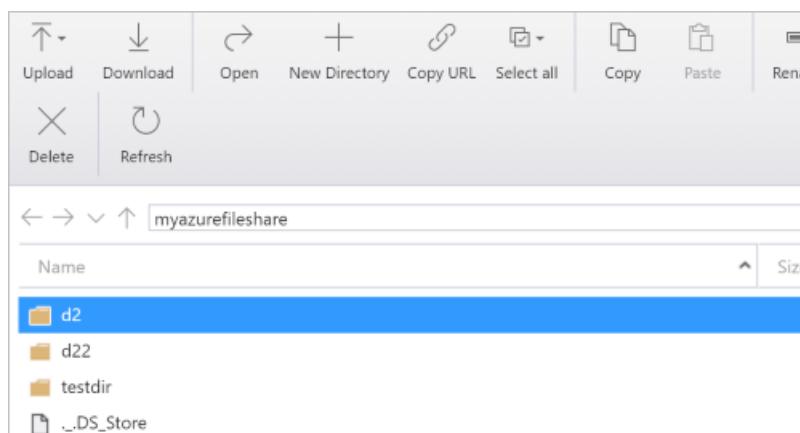
File shares contain files and folders (that can also contain files).

The following steps illustrate how to view the contents of a file share within Storage Explorer:-

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the file share you wish to view.
3. Expand the storage account's **File Shares**.
4. Right-click the file share you wish to view, and - from the context menu - select **Open**. You can also double-click the file share you wish to view.



5. The main pane will display the file share's contents.

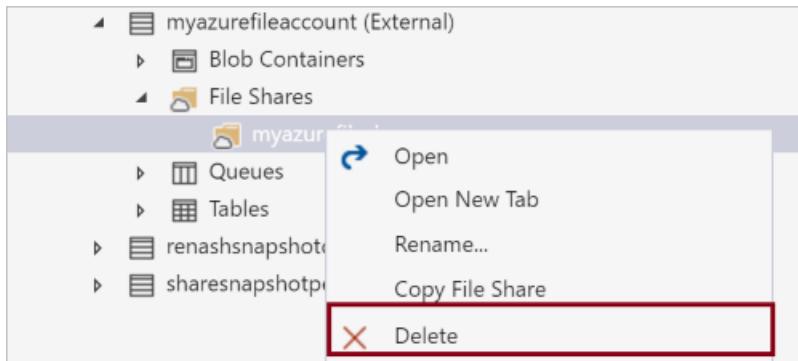


## Delete a file share

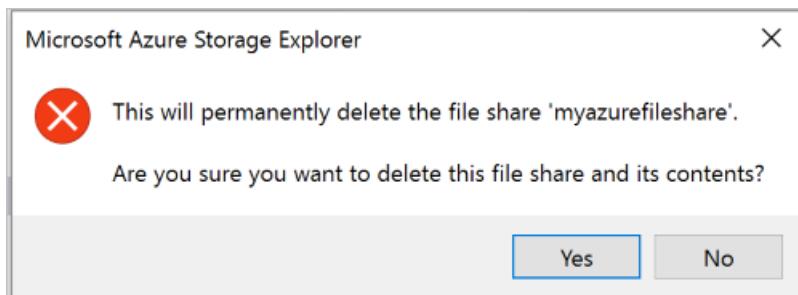
File shares can be easily created and deleted as needed. (To see how to delete individual files, refer to the section, [Managing files in a file share](#).)

The following steps illustrate how to delete a file share within Storage Explorer:

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the file share you wish to view.
3. Expand the storage account's **File Shares**.
4. Right-click the file share you wish to delete, and - from the context menu - select **Delete**. You can also press **Delete** to delete the currently selected file share.



5. Select **Yes** to the confirmation dialog.

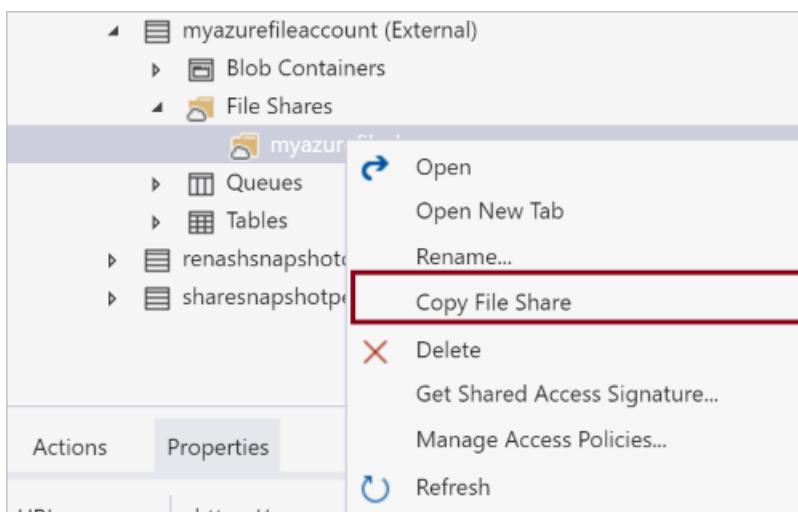


## Copy a file share

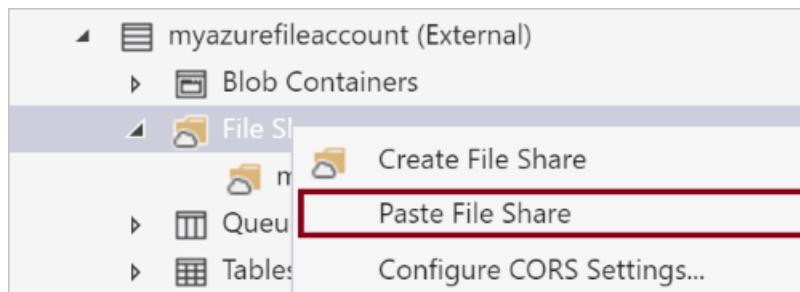
Storage Explorer enables you to copy a file share to the clipboard, and then paste that file share into another storage account. (To see how to copy individual files, refer to the section, [Managing files in a file share](#).)

The following steps illustrate how to copy a file share from one storage account to another.

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the file share you wish to copy.
3. Expand the storage account's **File Shares**.
4. Right-click the file share you wish to copy, and - from the context menu - select **Copy File Share**.



5. Right-click the desired "target" storage account into which you want to paste the file share, and - from the context menu - select **Paste File Share**.

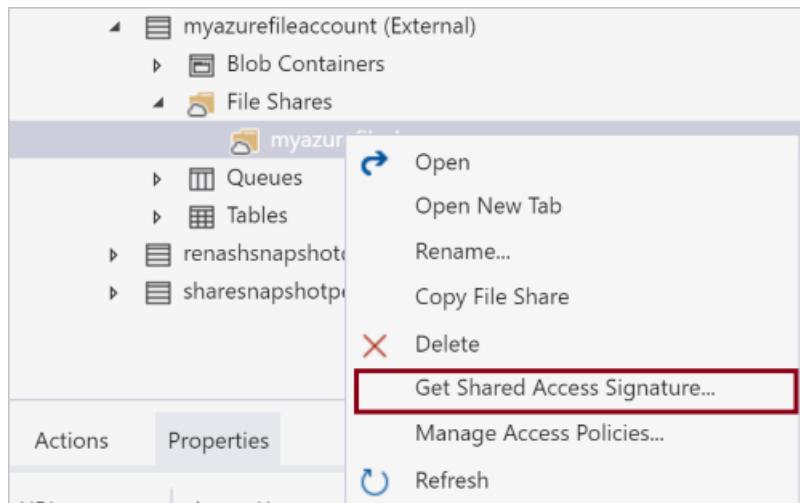


## Get the SAS for a file share

A [shared access signature \(SAS\)](#) provides delegated access to resources in your storage account. This means that you can grant a client limited permissions to objects in your storage account for a specified period of time and with a specified set of permissions, without having to share your account access keys.

The following steps illustrate how to create a SAS for a file share:+

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the file share for which you wish to get a SAS.
3. Expand the storage account's **File Shares**.
4. Right-click the desired file share, and - from the context menu - select **Get Shared Access Signature**.



5. In the **Shared Access Signature** dialog, specify the policy, start and expiration dates, time zone, and access levels you want for the resource.

## Shared Access Signature

Access policy:

Start time:

Expiry time:

Time zone:  
 Local  
 UTC

Permissions:

Read  
 Write  
 Delete  
 List

6. When you're finished specifying the SAS options, select **Create**.
7. A second **Shared Access Signature** dialog will then display that lists the file share along with the URL and QueryStrings you can use to access the storage resource. Select **Copy** next to the URL you wish to copy to the clipboard.

## Shared Access Signature

Share:

URL:

Query string:

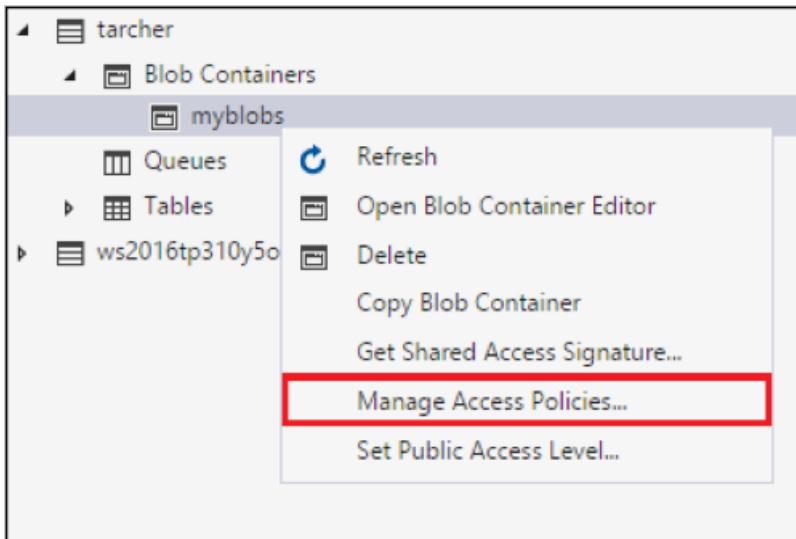
8. When done, select **Close**.

## Manage Access Policies for a file share

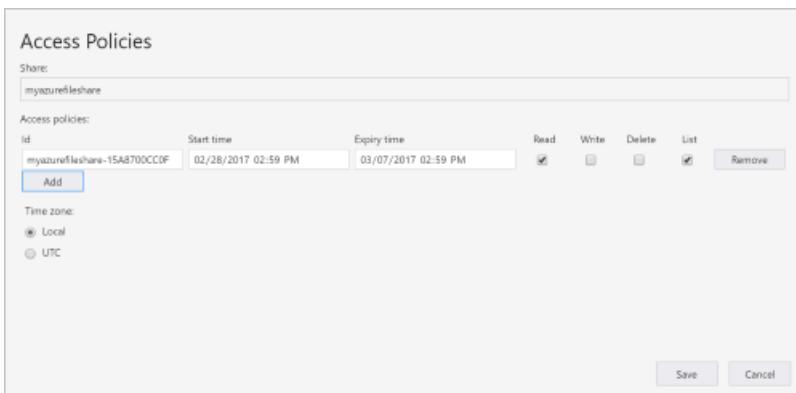
The following steps illustrate how to manage (add and remove) access policies for a file share: . The Access Policies is used for creating SAS URLs through which people can use to access the Azure Files resource during a defined period of time.

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the file share whose access policies you wish to manage.

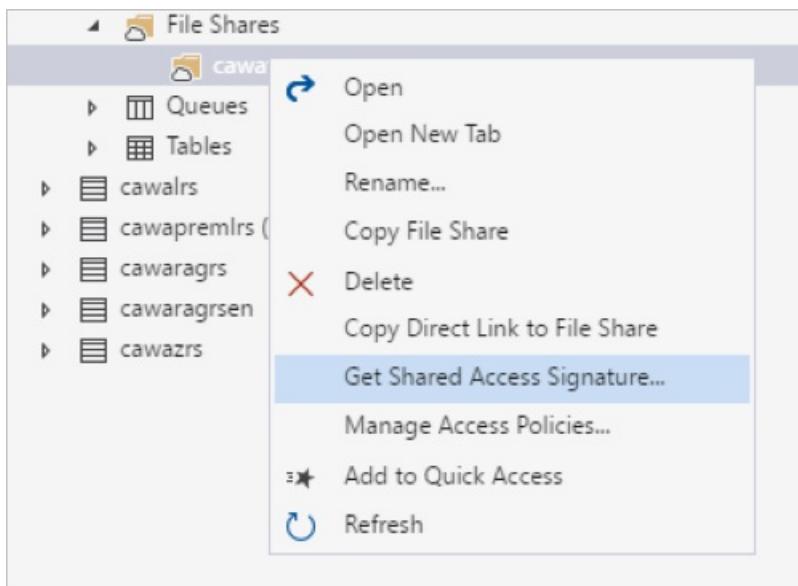
3. Expand the storage account's **File Shares**.
4. Select the desired file share, and - from the context menu - select **Manage Access Policies**.



5. The **Access Policies** dialog will list any access policies already created for the selected file share.



6. Follow these steps depending on the access policy management task:
  - **Add a new access policy** - Select **Add**. Once generated, the **Access Policies** dialog will display the newly added access policy (with default settings).
  - **Edit an access policy** - Make any desired edits, and select **Save**.
  - **Remove an access policy** - Select **Remove** next to the access policy you wish to remove.
7. Create a new SAS URL using the Access Policy you created earlier:



### Shared Access Signature

Access policy: myazurefileshare-15A8700CC0F

Start time: 03/06/2017 02:52 PM

Expiry time: 03/13/2017 02:52 PM

Time zone:  
 Local  
 UTC

Permissions:

Read  
 Write  
 Delete  
 List

**Create** **Cancel**

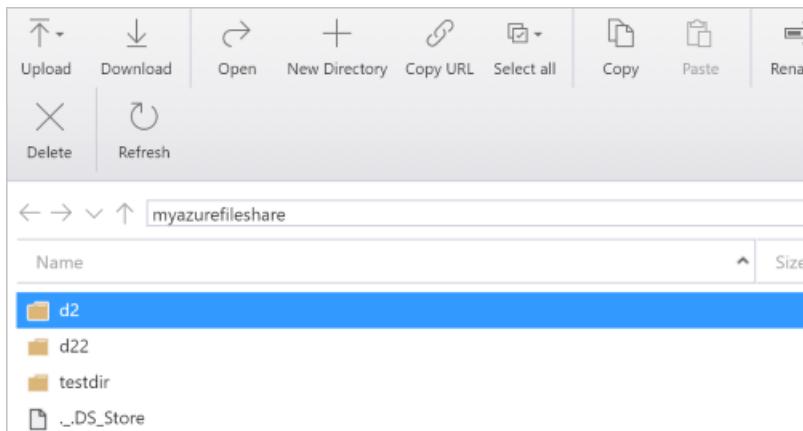
This is a configuration dialog for creating a Shared Access Signature. It includes fields for the access policy name, start and expiry times, time zone (set to Local), and a list of permissions (Read, Write, Delete, List) with checkboxes. At the bottom are 'Create' and 'Cancel' buttons.

## Managing files in a file share

Once you've created a file share, you can upload a file to that file share, download a file to your local computer, open a file on your local computer, and much more.

The following steps illustrate how to manage the files (and folders) within a file share.

1. Open Storage Explorer.
2. In the left pane, expand the storage account containing the file share you wish to manage.
3. Expand the storage account's **File Shares**.
4. Double-click the file share you wish to view.
5. The main pane will display the file share's contents.

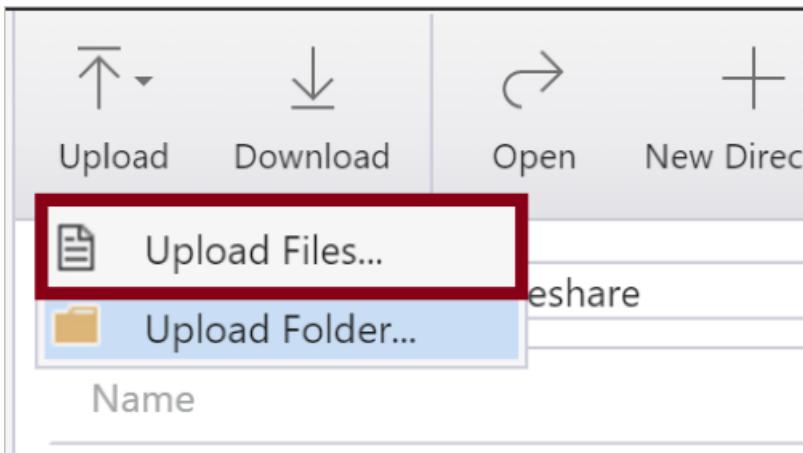


6. The main pane will display the file share's contents.

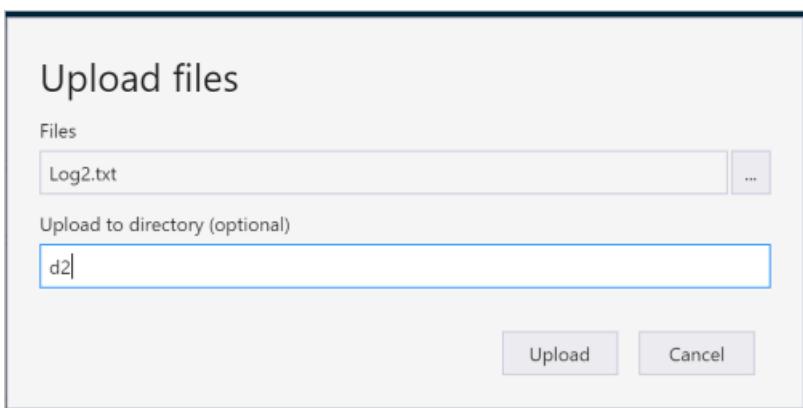
7. Follow these steps depending on the task you wish to perform:

- **Upload files to a file share**

a. On the main pane's toolbar, select **Upload**, and then **Upload Files** from the drop-down menu.



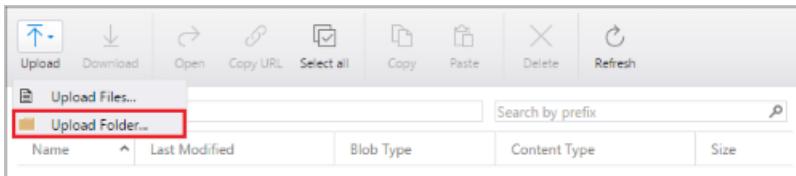
b. In the **Upload files** dialog, select the ellipsis (...) button on the right side of the **Files** text box to select the file(s) you wish to upload.



c. Select **Upload**.

- **Upload a folder to a file share**

a. On the main pane's toolbar, select **Upload**, and then **Upload Folder** from the drop-down menu.



- b. In the **Upload folder** dialog, select the ellipsis (...) button on the right side of the **Folder** text box to select the folder whose contents you wish to upload.
  - c. Optionally, specify a target folder into which the selected folder's contents will be uploaded. If the target folder doesn't exist, it will be created.
  - d. Select **Upload**.
- **Download a file to your local computer**
    - a. Select the file you wish to download.
    - b. On the main pane's toolbar, select **Download**.
    - c. In the **Specify where to save the downloaded file** dialog, specify the location where you want the file downloaded, and the name you wish to give it.
    - d. Select **Save**.
  - **Open a file on your local computer**
    - a. Select the file you wish to open.
    - b. On the main pane's toolbar, select **Open**.
    - c. The file will be downloaded and opened using the application associated with the file's underlying file type.
  - **Copy a file to the clipboard**
    - a. Select the file you wish to copy.
    - b. On the main pane's toolbar, select **Copy**.
    - c. In the left pane, navigate to another file share, and double-click it to view it in the main pane.
    - d. On the main pane's toolbar, select **Paste** to create a copy of the file.
  - **Delete a file**
    - a. Select the file you wish to delete.
    - b. On the main pane's toolbar, select **Delete**.
    - c. Select **Yes** to the confirmation dialog.

## Next steps

- View the [latest Storage Explorer release notes and videos](#).
- Learn how to [create applications using Azure blobs, tables, queues, and files](#).

# Azure Storage Explorer support lifecycle and policy

8/22/2022 • 3 minutes to read • [Edit Online](#)

This document covers the support lifecycle and policy for Azure Storage Explorer.

## Update schedule and process

Azure Storage Explorer is released four to six times a year. Microsoft may also bypass this schedule to release fixes for critical issues.

Storage Explorer checks for updates every hour, and downloads them when they're available. User acceptance is required, however, to start the install process. If users choose to update at a different time, they can manually check for update. On Windows and Linux, users can check for updates by selecting **Check for Updates** from the **Help** menu. On macOS, **Check for Updates** can be found under the **app** menu. Users may also directly go to the [Storage Explorer](#) download page to download and install the latest release.

We strongly recommend to always use the latest versions of Storage Explorer. If an issue is preventing you from updating to the latest version of Storage Explorer, open an issue on our [GitHub](#).

## Support lifecycle

Storage Explorer is governed by the [Modern Lifecycle Policy](#). It's expected that users keep their installation of Storage Explorer up to date. Staying up to date ensures that users have the latest capabilities, performance enhancements, security, and service reliability.

Starting with version 1.14.1, any Storage Explorer release that is greater than 12 months old will be considered out of support. All releases before 1.14.1 will be considered out of support starting on July 14, 2021. Versions that are out of support are no longer guaranteed to work fully as designed and expected. For a list of all releases, their release date, and their end of support date, see [Releases](#).

Starting with version 1.13.0, an in-app alert may be displayed once a version is approximately one month away from being out of support. The alert encourages users to update to the latest version of Storage Explorer. Once a version is out of support, the in-app alert may be displayed on each start-up.

## Releases

This table describes the release date and the end of support date for each release of Azure Storage Explorer.

STORAGE EXPLORER VERSION	RELEASE DATE	END OF SUPPORT DATE
v1.25.0	August 3, 2022	August 3, 2023
v1.24.3	June 21, 2022	June 21, 2023
v1.24.2	May 27, 2022	May 27, 2023
v1.24.1	May 12, 2022	May 12, 2023
v1.24.0	May 3, 2022	May 3, 2023
v1.23.1	April 12, 2022	April 12, 2023

STORAGE EXPLORER VERSION	RELEASE DATE	END OF SUPPORT DATE
v1.23.0	March 2, 2022	March 2, 2023
v1.22.1	January 25, 2022	January 25, 2023
v1.22.0	December 14, 2021	December 14, 2022
v1.21.3	October 25, 2021	October 25, 2022
v1.21.2	September 28, 2021	September 28, 2022
v1.21.1	September 22, 2021	September 22, 2022
v1.21.0	September 8, 2021	September 8, 2022
v1.20.1	July 23, 2021	July 23, 2022
v1.20.0	June 25, 2021	June 25, 2022
v1.19.1	April 29, 2021	April 29, 2022
v1.19.0	April 15, 2021	April 15, 2022
v1.18.1	March 4, 2021	March 4, 2022
v1.18.0	March 1, 2021	March 1, 2022
v1.17.0	December 15, 2020	December 15, 2021
v1.16.0	November 10, 2020	November 10, 2021
v1.15.1	September 2, 2020	September 2, 2021
v1.15.0	August 27, 2020	August 27, 2021
v1.14.2	July 16, 2020	July 16, 2021
v1.14.1	July 14, 2020	July 14, 2021
v1.14.0	June 24, 2020	July 14, 2021
v1.13.1	May 18, 2020	July 14, 2021
v1.13.0	May 1, 2020	July 14, 2021
v1.12.0	January 16, 2020	July 14, 2021
v1.11.2	December 17, 2019	July 14, 2021
v1.11.1	November 20, 2019	July 14, 2021

STORAGE EXPLORER VERSION	RELEASE DATE	END OF SUPPORT DATE
v1.11.0	November 4, 2019	July 14, 2021
v1.10.1	September 19, 2019	July 14, 2021
v1.10.0	September 12, 2019	July 14, 2021
v1.9.0	July 1, 2019	July 14, 2021
v1.8.1	May 10, 2019	July 14, 2021
v1.8.0	May 2, 2019	July 14, 2021
v1.7.0	March 5, 2019	July 14, 2021
v1.6.2	January 8, 2019	July 14, 2021
v1.6.1	December 18, 2018	July 14, 2021
v1.6.0	December 4, 2018	July 14, 2021
v1.5.0	October 29, 2018	July 14, 2021
v1.4.4	October 15, 2018	July 14, 2021
v1.4.2	September 24, 2018	July 14, 2021
v1.4.1	August 28, 2018	July 14, 2021
v1.3.1	July 11, 2018	July 14, 2021
v1.2.0	June 12, 2018	July 14, 2021
v1.1.0	May 9, 2018	July 14, 2021
v1.0.0 (and older)	April 16, 2018	July 14, 2021

# Storage Explorer Accessibility

8/22/2022 • 2 minutes to read • [Edit Online](#)

## Screen Readers

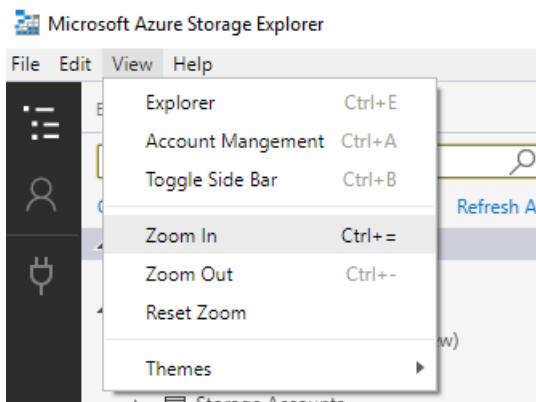
Storage Explorer supports the use of a screen reader on Windows and Mac. The following screen readers are recommended for each platform:

PLATFORM	SCREEN READER
Windows	NVDA
Mac	Voice Over
Linux	(screen readers are not supported on Linux)

If you run into an accessibility issue when using Storage Explorer, please [open an issue on GitHub](#).

## Zoom

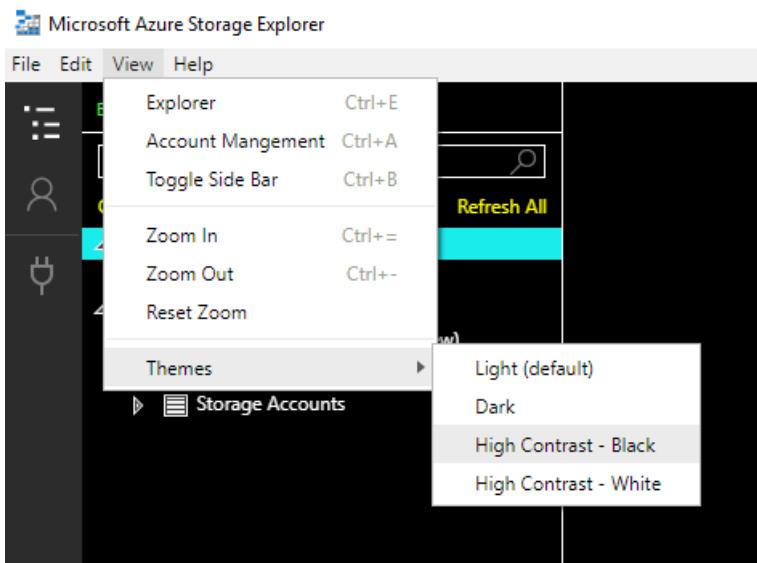
You can make the text in Storage Explorer larger via zooming in. To zoom in, click on **Zoom In** in the Help menu. You can also use the Help menu to zoom out and reset the zoom level back to the default level.



The zoom setting increases the size of most UI elements. It is recommended to also enable large text and zoom settings for your OS to ensure that all UI elements are properly scaled.

## High Contrast Themes

Storage Explorer has two high contrast themes, **High Contrast Light** and **High Contrast Dark**. You can change your theme by selecting in from the Help > Themes menu.



The theme setting changes the color of most UI elements. It is recommended to also enable your OS' matching high contrast theme to ensure that all UI elements are properly colored.

## Shortcut Keys

### Window Commands

COMMAND	KEYBOARD SHORTCUT
New Window	Control+Shift+N
Close Editor	Control+F4
Quit	Control+Shift+W

### Navigation Commands

COMMAND	KEYBOARD SHORTCUT
Focus Next Panel	F6
Focus Previous Panel	Shift+F6
Explorer	Control+Shift+E
Account Management	Control+Shift+A
Toggle Side Bar	Control+B
Activity Log	Control+Shift+L
Actions and Properties	Control+Shift+P
Current Editor	Control+Home
Next Editor	Control+Page Down
Previous Editor	Control+Page Up

## Zoom Commands

COMMAND	KEYBOARD SHORTCUT
Zoom In	Control+=
Zoom Out	Control+-

## Blob and File Share Editor Commands

COMMAND	KEYBOARD SHORTCUT
Back	Alt+Left Arrow
Forward	Alt+Right Arrow
Up	Alt+Up Arrow

## Editor Commands

COMMAND	KEYBOARD SHORTCUT
Copy	Control+C
Cut	Control+X
Paste	Control+V
Refresh	Control+R

## Other Commands

COMMAND	KEYBOARD SHORTCUT
Toggle Developer Tools	F12
Reload	Alt+Control+R

